Reconstruction of the MOS 6502 on the Cyclone II FPGA

TEAM Double O Four

Yu Chen (yc2615)
Jaebin Choi (jc3797)
Arthy Sundaram (as4304)
Anthony Erlinger (afe2104)

Reconstruction of the MOS 6502 on the Cyclone II FPGA

Table of Contents:

1. Divide and Conquer: Unit tests to Integration tests
2. Testing the ISA
3. Synthesis on the Cyclone II
4. Known bugs/issues

V. Application

1. Writing to the VGA Frame buffer
2. The Bouncing Ball

V. Conclusion

1. Lessons learned
2. Future direction
3. References
4. Appendix (Timing Diagrams, Adaptations)

Reconstruction of the MOS 6502 on the Cyclone II FPGA

Abstract

Owing to its low cost and simple yet powerful instruction set architecture the MOS 6502 processor is one of the best selling processing microprocessors in history. Here we have modeled and reconstructed a synthesizable 8-bit MOS 6502 processor in VHDL fully synthesizable on the Altera DE2 FPGA board. In recreating the 6502 many design considerations and modifications needed to be made to the original design in order to make it fully compatible with the more modern Cyclone II FPGA utilized on the Altera DE2. Our design differs in the two aspects that it (1) uses only a single phase clock (as opposed to the non-overlapping two phase clock used in the original) and (2) uses edge triggered D flip flops for internal registers in contrast to level sensitive latches which were used in the original design. Our design includes a basic test interface on the DE2 board in addition as well as integration test and simple ROM programs to test functionality. The process of creating our design consisted of two major sections. First is a thorough analysis of the architecture around the 6502 processor and the motivations behind the original design. Secondly, we will report on the process of designing of our own 6502 from design to implementation and testing.

I. Introduction: History and Significance of the 6502

Preface:

The Mos 6502 CPU has earned a prominent place in computing history. Owing to its versatility and low cost, the 6502 has been implemented in a myriad of devices including the Apple II, the Commodore 64, and the original Nintendo Entertainment System. Along with its contemporary, the Zilog Z80, the 6502 was largely responsible for the growth of computer games and the early operating systems in the 1980s. Today it remains in production within various peripherals and legacy devices. However, perhaps more relevant is the influence of the 6502's design on more modern. This arises from the fact that the 6502 was the first CPU to utilize a "reduced" instruction set with an 8 bit opcode. Today, many consider the 6502 to be the spiritual predecessor to the multiprocessor with interlocking pipelined stages (MIPS) CPU, which, in turn inspired the development of the ARM instruction set implemented in virtually every mobile device.

Why the 6502?

In the concept phase of this project our group threw around many ideas. Virtually all of these were half-baked, inconsistent, and posed a problem that was not relevant or tractable for this final project. After much discussion and debate we narrowed down the scope of this project to focus on the hardware/software interface through the design of an existing CPU. Again, we threw around many different ideas and after spending a great amount of time researching. We considered other famous processors such as Intel's 4004 and 8080, Motorola's 6800, and the Zilog Z80. However, there is a tremendous amount of community support in the "white hat" hacker community that has deconstructed and reverse engineered the original design from the instruction set down to the transistor. After further research it was clear that

there weren't any proprietary CPUs that are as well understood and documented as the MOS 6502 processor and after a discussion with the professor, we settled on our choice on the 6502.

Before we proceed further to our implementation details, we thought it deserves to quote its history from websource and the influence the 6502 had during its period and which inspired us to implement this processor as part of this class project.

> *The 6502 processor not only brought great influences to computers market, it developed the video game console, too. Most of the video game consoles uses refined version of 6502 processors. The first video game console which uses 6502 technology was the Atari 2600. Atari 2600 uses a simplified version of 6502, 6507 which can only address 8KB memory. (What is the source for this?)*

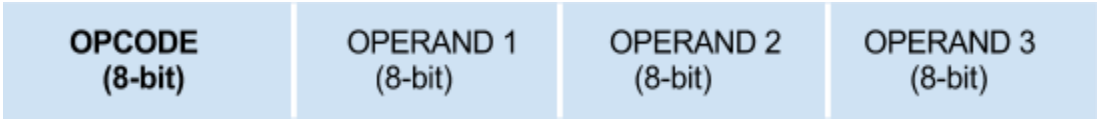
II. Understanding the Design of the 6502

Instruction Set Architecture

All instructions running on the 6502 are encoded within an 8-bit space which map to unique 62 operation codes (known as "opcodes") available to the processor. Each operation generally falls into one of three categories: (1) Arithmetic operations such as ADD, OR, XOR and so on. (2) Memory operations which store and retrieve data to and from system memory and the processor's internal registers. These include operations that load data into a register, transfer data between registers and (3) control flow operations which are used to allow a program to jump or branch to another location of execution within a running program. Generally speaking, control opcodes are the most complex in their execution as they involve many cycles across various components of the processor.

Opcodes:

Most operations (except for implied instructions such as *CLC*, *TAX*, Set processor status, etc…) accept up to three operands, stored in little endian format, which occupy the following adjacent bits in memory.



Since the processor can only fetch a single byte of data per clock cycle, an internal state machine is used to control when and how the input pipeline fetches data from memory.

Last 4 Bits:

First 4 Bits:	0000	0001	0010	0100	0101	0110	1000	1001	1010	1100	1101	1110
0000	BRK#	ORA(d,X)			ORAd	ASLd	PHP	ORA#	ASLA		ORAa	ASLa
0001	BPLr	ORA(d),Y			ORAd,X	ASLd,X	CLC	ORAa,Y			ORAa,X	ASLa,X
0010	JSRa	AND(d,X)		BITd	ANDd	ROLd	PLP	AND#	ROLA	BITa	ANDa	ROLa
0100	BMIr	AND(d),Y			ANDd,X	ROLd,X	SEC	ANDa,Y			ANDa,X	ROLa,X
0101	RTI	EOR(d,X)			EORd	LSRd	PHA	EOR#	LSRA	JMPa	EORa	LSRa
0110	BVCr	EOR(d),Y			EORd,X	LSRd,X	CLI	EORa,Y			EORa,X	LSRa,X
1000	RTS	ADC(d,X)			ADCd	RORd	PLA	ADC#	RORA	JMP(a)	ADCa	RORa
1001	BVSr	ADC(d),Y			ADCd,X	RORd,X	SEI	ADCa,Y			ADCa,X	RORa,X
1010		STA(d,X)		STYd	STAd	STXd	DEY		TXA	STYa	STAa	STXa
1100	BCCr	STA(d),Y		STYd,X	STAd,X	STXd,Y	TYA	STAa,Y	TXS		STAa,X	
1101	LDY#	LDA(d,X)	LDX#	LDYd	LDAd	LDXd	TAY	LDA#	TAX	LDYa	LDAa	LDXa
1110	BCSr	LDA(d),Y		LDYd,X	LDAd,X	LDXd,Y	CLV	LDAa,Y	TSX	LDYa,X	LDAa,X	LDXa,Y
1100	CPY#	CMP(d,X)		CPYd	CMPd	DECd	INY	CMP#	DEX	CPYa	CMPa	DECa
1101	BNEr	CMP(d),Y			CMPd,X	DECd,X	CLD	CMPa,Y			CMPa,X	DECa,X
1110	CPX#	SBC(d,X)		CPXd	SBCd	INCd	INX	SBC#	NOP	CPXa	SBCa	INCa
1111	BEQr	SBC(d),Y			SBCd,X	INCd,X	SED	SBCa,Y			SBCa,X	INCa,X

Memory Addressing Modes

Although there are only 62 operation types in the 6502, many can access or modify memory depending on how the programmer wishes to access memory. For instance, the *load accumulator* (*LDA*) operation, which takes a value from memory and stores it in the processors internal accumulator register, can be called with an immediate, zero page, absolute (X or Y), indirect (X or Y).

There are a total of 13 addressing modes:

- Implicit (1 byte instructions)
- Accumulator (1 byte instructions)
- Immediate (2 byte instructions)
- Zero page (2 byte instructions)
- Zero page, X (3 byte instructions)
- Zero page, Y (3 byte instructions)
- Relative (3 byte instructions)
- Absolute - (3 byte instructions)
- Absolute, X - (3 byte instructions)
- Absolute, Y - (3 byte instructions)
- Indirect, - (3 byte instructions)
- Indexed Indirect - (2 byte instructions)
- Indirect Indexed - (2 byte instructions)

There exist three registers which are used temporarily store data between operations. These are the accumulator (*ACC*), and the *X* and *Y* registers. Values in these registers can be intermittently loaded, incremented, fetched, or stored back into memory.

Opcode Bit Format

All instructions are encoded in a particular format which allows them to be most efficiently decoded after they are fetched from the data bus from memory. Upon thorough examination, much of the internal architecture of the 6502 can

be inferred from its instruction set architecture. Therefore, fully understanding the instruction set architecture is essential to fully understand the full CPU.

Depending on an instruction's role, its eight bits can be categorized according to a format of **AAABBBCC**. The first three and last two bits (i.e. **AAA**---**CC**) are used in identifying the type of opcode. The middle three bits (i.e. ---**BBB**--) are used to identify the addressing mode according to the below table.

BBB	Addressing Mode
000	(zero page,X)
001	zero page
010	#immediate
011	absolute
100	(zero page),Y
101	zero page,X
110	absolute,Y
111	absolute,X

Addressing modes not in this table are implied by their operation or occupy additional bits than just BBB. For instance, the addressing mode on increment X instruction (INX), is always implied (just add +1 to the X register); the addressing mode on a branch instruction is always relative, and so on. A good example of an instruction that uses multiple addressing modes can be seen with the *load accumulator* instruction (LDA).

```
MODE            SYNTAX       HEX LEN TIME
Immediate      LDA #$44      $A9  2   2
Zero Page      LDA $44       $A5  2   3
Zero Page,X    LDA $44,X     $B5  2   4
Absolute       LDA $4400     $AD  3   4
Absolute,X     LDA $4400,X   $BD  3   4+
Absolute,Y     LDA $4400,Y   $B9  3   4+
Indirect,X     LDA ($44,X)   $A1  2   6
Indirect,Y     LDA ($44),Y   $B1  2   5+
```

Microarchitecture

[Stages of execution]

Predecode

The predecoder is combinational circuitry that pre-processes only the "opcode" as soon as it is fetched from the memory in T1 cycle. The pre-decoded opcode is then put onto the instruction register (IR) at the beginning of T2 and the timing generator also gets timing information from the predecoder and generates appropriate timing states for the current opcode in the IR.

Instruction register storage

IR, stores the opcode through the instruction execution cycle starting from T2 till the beginning of the next cyclic T2. The IR is implemented as a D-FlipFlop. As soon as the opcode is stored in the IR, the decoding and actual execution of the instruction begins.

Instruction decode and execution (Can happen concurrently)

The random control logic takes as input the opcode from the IR and the timing state from the timing control generator. In our implementation, the random control logic encompasses the sequential circuitry of decode and execution and the combinational ALU circuitry. Each opcode is tested for bit patterns or opcode equivalents and then the corresponding operation that needs to happen in each cycle is executed depending
on the "tcstate" from the timing controller. For example:

```
process(clk)
if ( risingedge(clk)) then
        if(opcode == XX and tcstate = T1) then
                AI<=databus;
                SUMS<=1;
           end if;
end if;
```

The above code tests for T1 as that's the tcstate value when T2 begins and loads the value from the data bus onto the AI register at the beginning of T2 and sets SUMS to 1. The ALU circuitry is combinational and responds to the state change and computes sum of the value with that in the BI register and *alucarryin* flag and stores the result in ADD register in the same cycle.

[Components]

Program counter (PC)

The program counter is incremented every clock cycle and the opcode and the following byte is fetched in two consecutive cycles. In the event the fetched opcode is single byte, the following fetched byte is ignored and sometimes, the PC is either decreased or retained depending on the opcode under execution.

Address bus registers (ABH and ABL)

We use a 8-bit ABL and 8-bit ABH registers to form the 16bit required to address the memory. The memory is asynchronous hence once address is put on these registers at the rising edge of current cycle, before the next rising edge the data will be available on the "databus". This register is write only, the processor can only write to the address bus registers.

Databus (di)

The databus input lines contain the data read from memory addressed by the registers ABH and ABL when the W_R is 1 (read). The databus is not a register and hence at the rising edge of the clock it will be either stored in the ACC, X, Y, AI, BI, SR, S registers depending on the opcode and the current tcstate.

Data output register (DOR)

The 8-bit data output register holds the data to be written to the memory in the current cycle. At the rising edge of clock the DOR takes the input written to it and when the W_R signal set to write, the value is stored into the memory to which the ABH and ABL registers are pointing to.

Stack pointer (S), index registers (X and Y), status registers (P), and accumulator (ACC)

They all can be written into and read from. Any addition and operation on content of these registers need to be done by the ALU and hence the values should be stored first onto the ALU input registers and the corresponding ALU flags must be set. The stack pointer is a 8-bit register holding the lower 8-bit of the stack address where the next push will store the data to. The stack grows from 0x1ff to 0x100. After every push operation PHA/PHP, the stack pointer points to the lower memory location and before every pull - PLA/PLP, the SP needs to be incremented and then a memory read is performed to fetch the value from the stack (pop).

The status registers, hold the status of the alu or store or register transfer operation performed from the current cycle. The semantics of this is same as that of the 6502 and can be found on the Appendix.

Timing Generator

Operation of the processor is further complicated by the fact that all instructions take multiple clock cycles depending on the addressing mode or conditions that can occur during execution such as a carry or page crossing being generated. The simplest instructions, such as LDA, STA and so on take a minimum of two cycles. More complex instructions such as those involving complex address modes or interrupt operations (e.g. JSR, RTI) can take up to 7 cycles depending if a carry is generated or a page crossing.

The current state of the processor is therefore stored through a Mealy state machine to represent the current state of the processor. At any given time, the processor can exist in one of ten possible states (T0-T5, T2+T0, VEC1, SD1-2). From our own analysis we have derived the state machine (which will be described in the next chapter).

The derived state machine

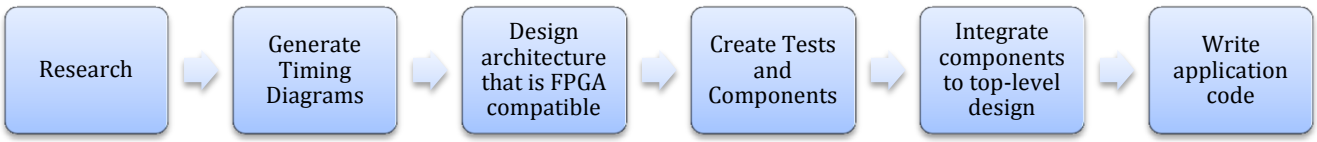
The processor is designed such that it responds to the positive levels of the two phase clocks, enabling memory transfers and instruction/opcode fetch to happen during the first phase and decoding and datapath transfers and ALU operations to happen over the second phase. The datapath is most certainly behind the control path - which handles the instruction fetch and decode - while the next opcode is being processed in the control plane, the last cycle of the previous instruction could still be in execution in the data path thereby achieving a very primitive form of pipelining.

II. Redesign of the 6502:

Understanding the entire structure of the 6502 formed the premise for the design of our own architecture. However, understanding the 6502 was an arduous task. Many hours were spent on formulating timing diagrams for each register, bus, and control lines. Challenges quickly arose that were not expected early on in the design process. Most setbacks were related to the fact that the 6502's design is over 30 years old and was built in an entirely different process technology from what is available today. To make matters worse, none of us fully understood the design constraints imposed by and FPGA architecture. Much of our early design was based on the assumption of a latch implementation. Somewhere around the three week mark, we realized our mistake of using tri-state buffers and high impedance switches as these are not necessarily stable. Thus, much redesign was done implementing multiplexers in lieu of a single bus being shared across many components. After about a month or so, we realized that memory data written to memory might not be available to the input within the same clock cycle.

At a more fundamental level, progress was stymied by the sheer complexity of the 6502's design. Although it is a very old and primitive processor by today's standards the cyclomatic complexity and number of possible independent states of the processor is still immense. Faced with this complexity it can be very easy to become overwhelmed by the scope of the task or become distracted by what is and is not important.

This created a "Big Design Up Front" (BDUF) problem in which we spent much effort making our understanding of the CPU. Every assumption was checked with everyone else yet persistent disagreements stymied progress before work could be started on a concrete implementation. Our design process evolved in many discrete stages:



Documents, timing diagrams, and other work created during the concept phase is shown in Appendix B.

The "whiteboard" representation of our circuit model went through three different implementations before a concrete design was implemented. This modified 6502 will henceforth be referred to as i6502 to identify as it is "inspired" from the original 6502. Our design process went something like this:

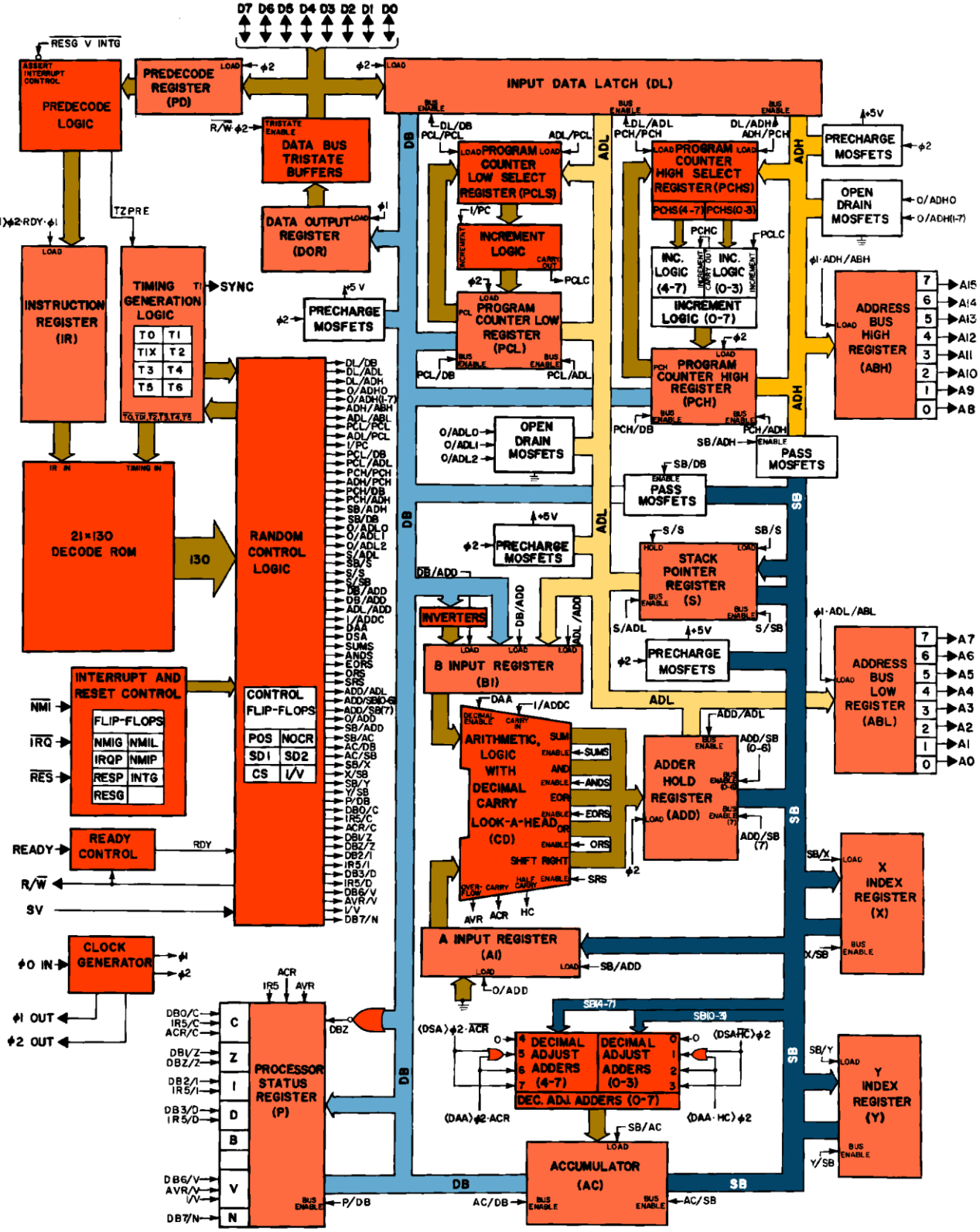
1.) Research 6502 architecture and build timing diagrams. **(2 months)**
2.) Mapping timing diagrams to state and control logic **(2 months)**
3.) Create and test individual components in VHDL
4.) Integrate and test high level structure
5.) Run test programs written in 6502 assembly

Generating Timing Diagrams

Reflecting back on the design process for this project, all of us would agree that the majority of our time was spent on timing diagrams. We started with some of the simpler instructions such as Load Accumulator (LDA), Store Accumulator (STA) and worked our way up to the more complex instructions such as *jump with subroutine* (JSR), *add with carry* (ADC), and so on. Many timing diagrams needed to be redone or later modified due to a misunderstanding or false assumptions about what aspects of the i6502 could be implemented on an FPGA.

Creating a design that is fully synthesizable on an FPGA

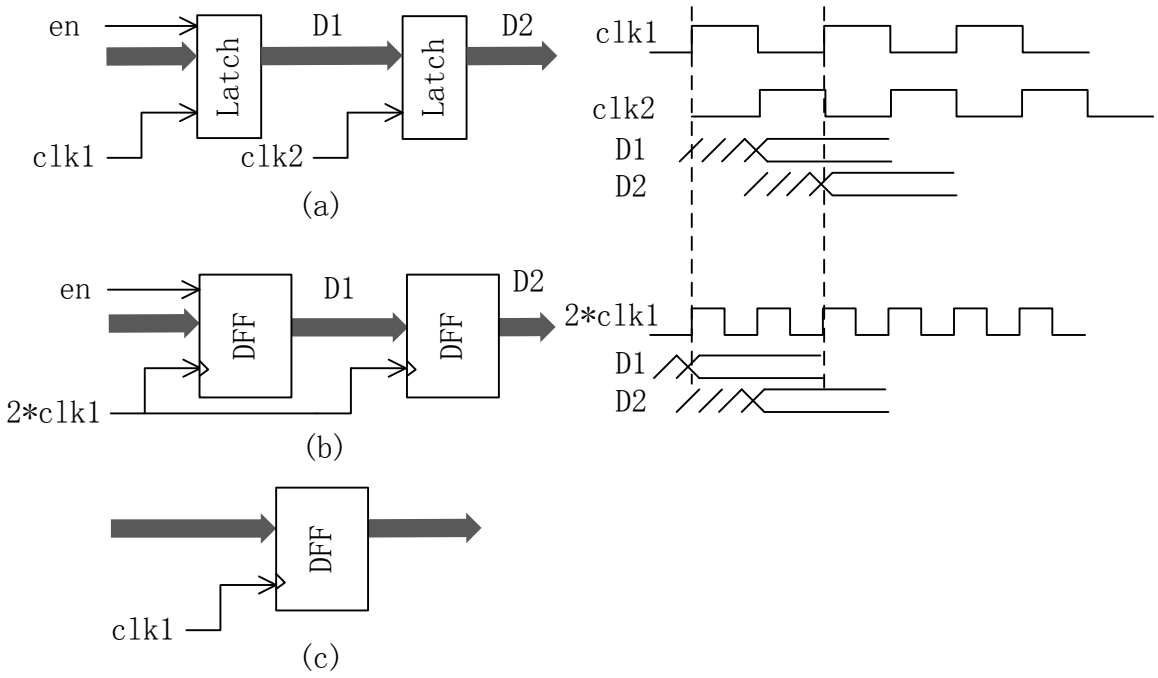
The original 6502 structure is represented in the following diagram:


D7 D6 D5 D4 D3 D2 D1 D0
RESG V INTG
ASSERT INTERRUPT CONTROL
PREDECODE LOGIC
PREDECODE REGISTER (PD)
INPUT DATA LATCH (DL)
TRISTATE ENABLE
DATA BUS TRISTATE BUFFERS
DATA OUTPUT REGISTER (DOR)
INSTRUCTION REGISTER (IR)
TIMING GENERATION LOGIC
SYNC
PRECHARGE MOSFETS
21×130 DECODE ROM
130
RANDOM CONTROL LOGIC
INTERRUPT AND RESET CONTROL
FLIP-FLOPS
NMIG NMIL
IRQP NMIP
RESP INTG
RESG
CONTROL FLIP-FLOPS
POS NOCR
SD1 SD2
CS I/V
READY CONTROL
CLOCK GENERATOR
PROGRAM COUNTER LOW SELECT REGISTER (PCLS)
PROGRAM COUNTER HIGH SELECT REGISTER (PCHS)
INCREMENT LOGIC
INC. LOGIC (4-7)
INC. LOGIC (0-3)
INCREMENT LOGIC (0-7)
PROGRAM COUNTER LOW REGISTER (PCL)
PROGRAM COUNTER HIGH REGISTER (PCH)
OPEN DRAIN MOSFETS
PASS MOSFETS
ADDRESS BUS HIGH REGISTER (ABH)
ADDRESS BUS LOW REGISTER (ABL)
STACK POINTER REGISTER (S)
INVERTERS
B INPUT REGISTER (BI)
ARITHMETIC, LOGIC WITH DECIMAL CARRY LOOK-A-HEAD (CD)
SHIFT RIGHT
ADDER HOLD REGISTER (ADD)
A INPUT REGISTER (AI)
X INDEX REGISTER (X)
Y INDEX REGISTER (Y)
DECIMAL ADJUST ADDERS (4-7)
DECIMAL ADJUST ADDERS (0-3)
DEC. ADJ. ADDERS (0-7)
PROCESSOR STATUS REGISTER (P)
ACCUMULATOR (AC)
DB
ADL
ADH
SB

However, the original design cannot be integrated in an FPGA for three reasons:

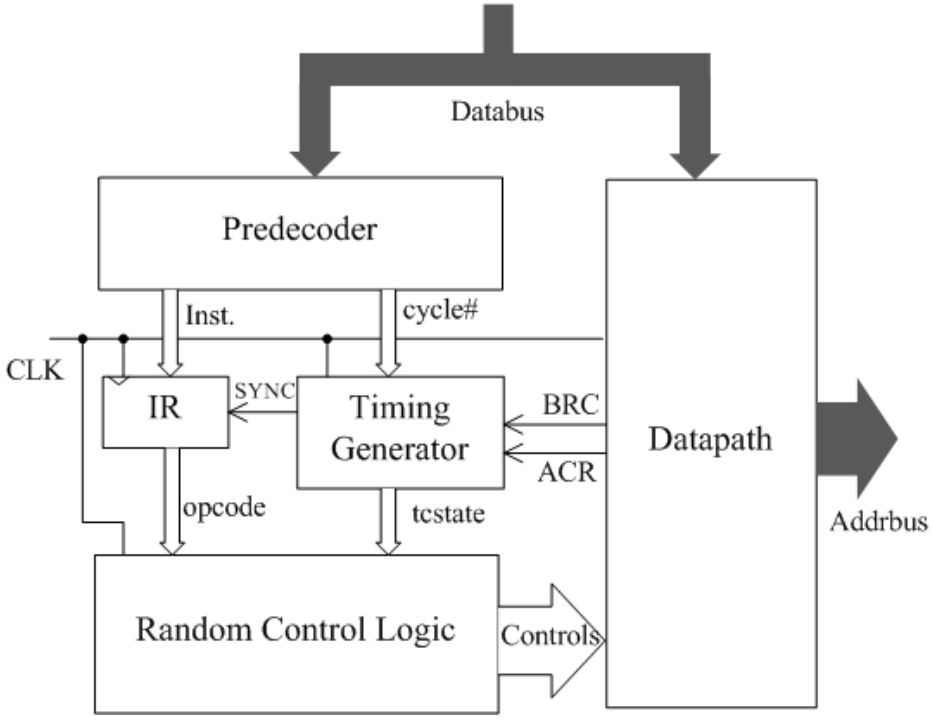
1. The 6502 uses a two phase non-overlapping clock, however, the DE2 has a single clock structure.
2. The 6502 uses latches which cannot be reliably synthesized on an FPGA. To overcome this issue, we had to redesign the register and memory structure to use flip-flops.
3. Tri-state buffers which are important in selecting how bus lines get asserted also cannot be synthesized on an FPGA.

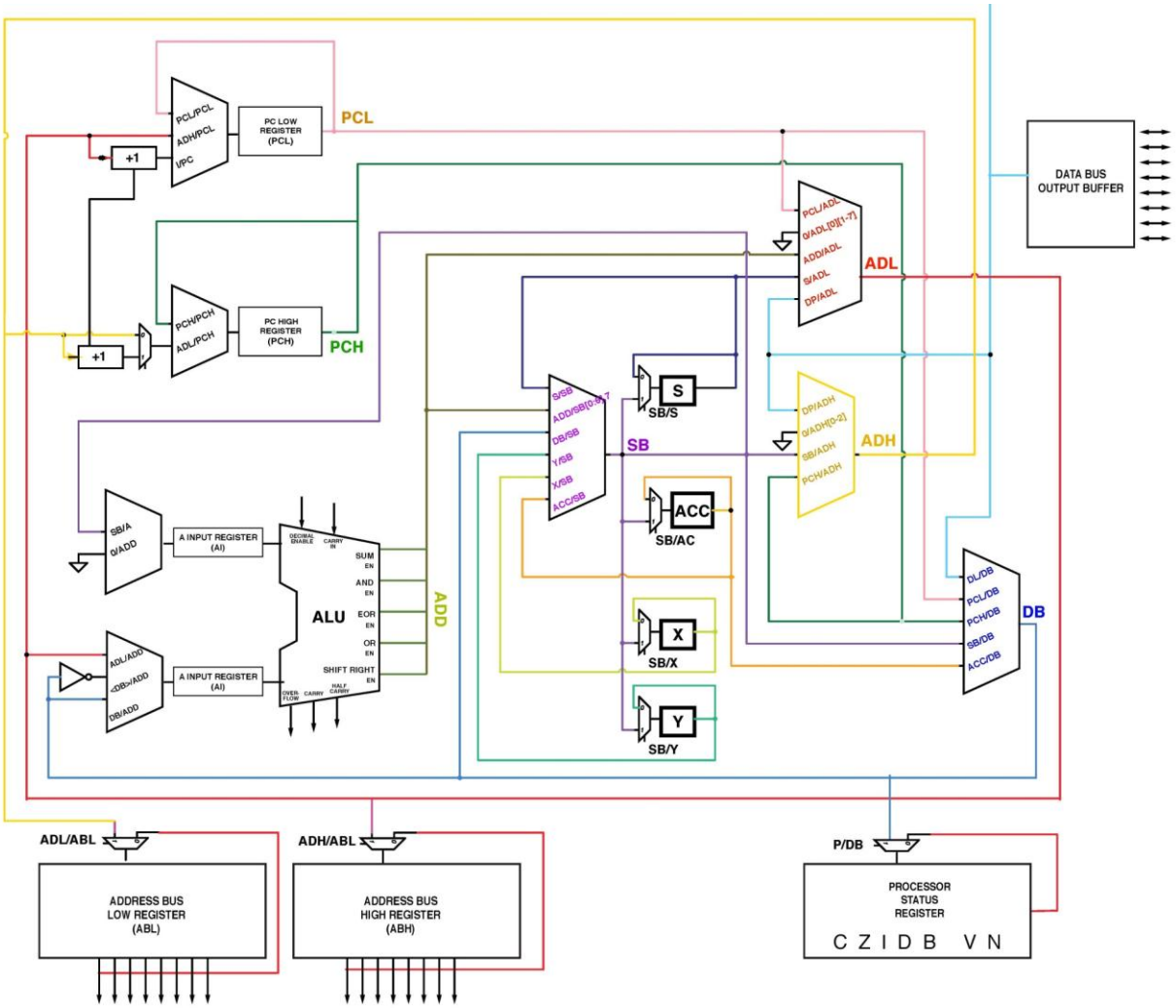
The synchronous-asynchronous problem



We thought it would be worth mentioning how we came to our decision of the single-clock implementation. The diagram above shows (a) a latch behavior (original 6502), (b) double-frequency clock implementation, and finally (c) a single clock implementation. Here we compare cases (b) and (c) in modeling the (a) behavior of the original 6502. Case (b) is straightforward: simply replace all latches with D-Flip-Flops. To match the (a) behavior, (b) is required a front-shift of all commands as observed in the timing diagram. Thus, there is a half-cycle shift between the two schemes. Case (c) which avoids using latches is based on our observation that two successive latches with a two-phase non-overlapping clock is actually a single DFF. And we also have found that all the latches inside the original 6502 appear in pairs. For example, in the control path, predecode and instruction latches are a pair; in the data path, data input latch and any one of X, Y or ACC latches are another pair, etc. In any single cycle time (full cycle), data must go through a pair of these latches. Combining these two observation, we came up with the second idea to use one DFF to replace two successive latches, which is also the final decision to enable us implement the process on FPGA.

With the single D-flip flop replacement of two latches, we can simplify design from the extremely complex two-phase 6502 to a simpler implementation, as shown blow. To our convenience, there is only one register in the control path. The data path also only contains single layer of register with a MUX in front of each to choose the correct driver at any given time, which are controlled by the random control logic.

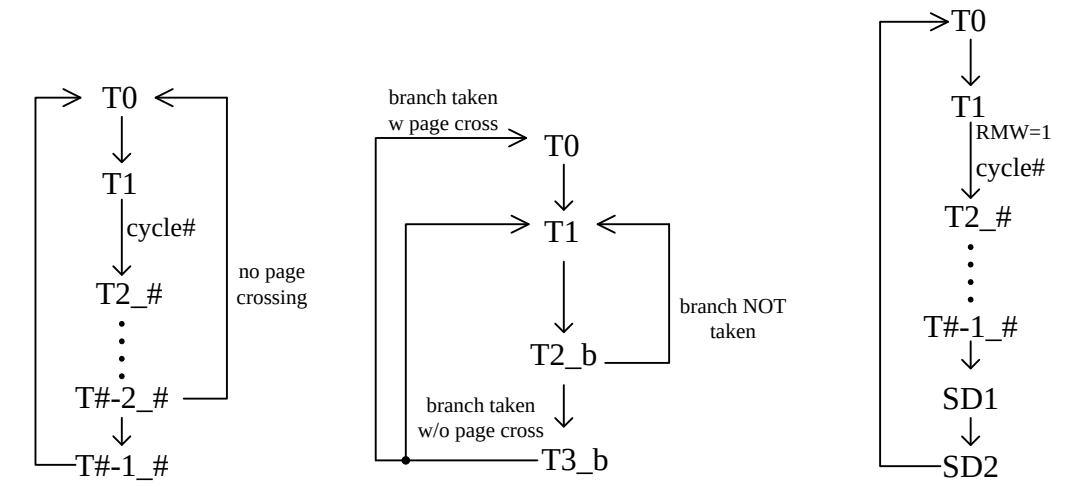




Redesigning the timing generator

We built the timing generator of the i6502 closely following the original 6502 design (please refer to previous sections). However, several modifications are made which eases our design a lot.

1) First of all, we divided all the opcodes into three sections: a) normal instructions b) branch instruction and c) read-modified-write instruction. Let's find the reasons from following:
   a) Normal instructions: They are the instructions which do not belong to the following two categories. Their timing states are very simple. It is almost linear with only one branch as you can see from the following figure. It starts at T0 and T1 in the first two cycles and keeps go on. The cycle# information from the predecoder tells the state machine when to return the starting point (T0). The only branch happens at the second to the last cycle T#-1_#. If there is no page crossing, it will skip the last state T#_# and go back to T0. Otherwise, it T#_# will be the next state.



(a) timing states for normal instruction

(b) timing states for branch instruction

(c) timing states for RMW instruction

b) Branch instruction: Most of the other instructions starts with T0 and come back to T0 after it finishes. However, branch may not. Three cases could happen with brach instruction: i) branch not taken (2 cycles); ii) branch taken without page crossing (3 cycles); iii) branch taken with page crossing (4 cycles). As we shown in the following figure, for the first two cases, it goes back to T1 after finishes, rather than T0. This is because, during the last cycle when judging either takes the branch or not, a new opcode has already been fetched on the databus. It doesn't need another T0 cycle to do the same thing again.

c) Read-modified-write instructions (RMW): These are the instructions which take the original data from memory, modify it according to the instruction and store it back into the same address. The share the same address modes with the other opcodes, but has two extra cycles. For example normal absolute address mode has 4 cycles, but RMW needs 6 cycles. The have the exactly same behaviors as the other address mode in the first several cycles, and the only distinct operations appear at the extra 2 cycles, which are labeled as SD1 and SD2. In SD1, corresponding arithmetic (e.g inc, dec, ror) will be conducted on the data; In SD2, the modified data will be stored back. In both cycles, the CPU are writing rather than reading the memory.

Mapping State and signal control

Knowing the facts that there are 152 instructions inside the 6502 and the longest opcode takes 7 cycles is really annoying in the beginning, as it may be an endless way to figure out what signal controls are generated in each cycle for any specific opcode (which is up to 152*7=1064 cases…). Fortunately, those brilliant designers of 6502 designed these 152 opodes in a way that many general patterns can be found. These patterns shrink the number of cases with different signal controls into less than 50 cases! Here is an example: an opocde with absolute address mode lasts for 4 cycles. However, no matter what function it is, the behaviors of the data path in the first two cycles are exactly the same, which means they all share the same signal control. More specifically, as you can see from the table below, in T2, the data path must load the data from memory into the ALU and add it with x"00" (the two operands are stored in AI and BI respectively); in T3, the data path must send the output of ALU from the previous cycle into ABL (lower 8 bits of the address bus) and load the new data from memory into ABH (higher 8 bits of the address bus).

Address Mode: Absolute			
	T2	T3	T0(next opcode)
ADDR	1002(LL)	1003(HH)	HHLL
DATA	LL	HH	??
PC	1003	1004	1005
IR	opcode 1	opcode 1	opcode 2
RTL operations:	PC<=PC+1	PC<=PC	
	ABL<=PCL	**ABL<=ADD**	
	ABH<=PCH	**ABH<=Data**	
	AI<=00	Sums<='0'	
	BI<=Data		
	Sums<='1'		

Since there are 24 opcodes have absolute address mode, the original 24*2 individual cases can now be shared by two cases mentioned above. 24X of effort are saving in this example.

With this in mind, we believe it is crucial to understand the general pattern of timing diagrams, control paths, and enable which dictate which components of the processor are active at any given time. To accomplish this, information was gathered from the aforementioned community resources [REFERENCES] and compiled into a single spreadsheet containing exactly 712 rows for all of the 152 cycle-complete instructions. An excerpt showing a very small subsection of this spreadsheet is shown below:

Mnem	Addr. Mode	Opcode	Orig	New	# Bytes	N	V	B	D	I	Z	C	Timi	Done	Addr_Op	Din_Le	Rd_En	ALU_OUT	PD	DL	DOR	PC_Op	SP_Op	A	B	ALU_OP	A_Le	X_Le	Y_Le	PS	BIN	TIMING
													6																				110
INX	IMP	E8	2	2	1	N					Z		0	done										x_reg	one	add		le		nz	11101000	000
													1																				001
INY	IMP	C8	2	2	1	N					Z		0	done										y_reg	one	add			le	nz	11001000	000
													1																				001
JMP	ABS	4C	3	5	3								0		pc_p	en	read					inc									01001100	000
													1			en			din			inc										001
													2									split										010
													3	done																		011
													4																				100
JMP	(IND)	6C	5	8	3								0		pc_p	en	read					inc									01101100	000
													1			en			din													001
													2		split		read															010
													3		split_p	en	read															011
													4			en			din													100
													5									split										101
													6	done																		110
													7																			111
JSR	ABS	20	6	5	3								0		pc_p	en	read					inc									00100000	000
													1		sp	en		pch	din				push									001
													2		sp			pcl				split	push									010
													3	done																		011
													4																			100
LDA	ABS	AD	4	5	3	N					Z		0		pc_p	en	read					inc									10101101	000
													1			en			din			inc										001
													2		split		read															010
													3	done		en																011
													4												din	passB	le			nz		100
LDA	ABS,X	BD	4	5	3	N					Z		0		pc_p	en	read					inc									10111101	000

From this data, certain distinct patterns arise:

1. Recall that we call the last two bits of opcodes cc. From the study, we found any opcodes with cc=10 are conducting operations on the accumulator register; any opcodes with cc=01 are conducting operations on the X index register; any opcodes with cc=10 are conducting operations on the Y index register. There are no opcodes which end in “11”

2. Any instructions shared with the same address mode, they will have the exact same behaviors in the datapath in any cycles except T0 and T1 (for RMW opcodes, SD1 and SD2 have different behaviors too). For example, LDA, LDY, LDX, INC (absolute) have the same behaviors as we shown in the above table.
3. Any opcodes, no matter what their address modes are, share the same behaviors in T0 and T1. For example, LDA with absolute, zero page, zero page indirect and etc share the same data path operations. More specifically, as we shown in the table below, the data fetched from memory will always be sent into the accumulator in T0; in T1, it prepares for the next instruction.

Instructions: LDA Address Mode: Don't care		
	T0	T1
ADDR	1002(LL)	1003(HH)
DATA	LL	HH
PC	1003	1004
IR	opcode 1	opcode 1
RTL operations:	PC<=PC+1	PC<=PC+1
	ABL<=PCL	ABL<=PCL
	ABH<=PCH	ABH<=PCH
	ACC<=Data	

4. Each instruction takes a minimum of two cycles but no more than 7 cycles and therefore the timing state can be represented within a 3 bit register.
5. The ALU is only used once per instruction.
6. 13 total modes of the ALU. Can represent the ALU operation in 4 bits.
7. There are many operations where the ALU loads zeros into on register, to pass the value of register A to its output.

Work distribution:

Actually, these patterns not only help us to understand the operation inside the process, but also easy our life when we distribute our work when implementing the instructions operations in VHDL. We divided the total 152 opcodes into four parts:

1) Opcodes end with 10 (accumulator operation);
2) Opcodes end with 01 (index register X operation);
3) Opcdoes end with 00 expect 4) and 5) (index register Y operations);
4) All the branch and jump instructions;
5) All the single byte instructions.

Writing and Testing VHDL Code of the processor

The VHDL code of the process (CPU core) is entirely pattern based. If certain pattern is matched, we ask the process to do the corresponding operations. Thus, one can imagine that the CPU.vhdl will be composed with a huge list of if…else statement. Here is an example code in T2 of absolute address mode:

```
--Address Mode: Absolute; aaa: don't care; cc: don't care.
```

```
--Timing: T2
      if (opcode(4 downto 2)="011" and tcstate(2)='0' ) then
            PC<=PC+1;
            ABL<=std_logic_vector(PC(7 Downto 0))
            ABH<=std_logic_vector(PC(15 Downto 8));
            Sums<='1';
            AI<=x"00";
            BI<=unsigned(Databus);
            I_ADDC<='0';
            end if;
```

In the if conditions, opcode(4 downto 2)="011" will be matched with all the opcodes with absolute address mode. tcstate(2)='0' will be met when the system enters T2 cycle. With both conditions, the operations followed will be active.

High Level Implementation on the Altera DE2 Board:

Our i6502 system connects to an off-chip asynchronous memory via an 8-bit address bus and a 16-bit address bus with a Read/Write line indicating whether the operation is read or write. The program to be executed will be stored in the SRAM and the i6502 will be pre-configured / boot loaded with the program start address. The processor operates on a single clock and responds to positive edge triggers of the clock. The reset pin can be used to reset the processor and restart execution of the program in memory.
The program should comply with i6502 instruction set architecture described in detail below.

IV. Testing

1. Testing the ISA

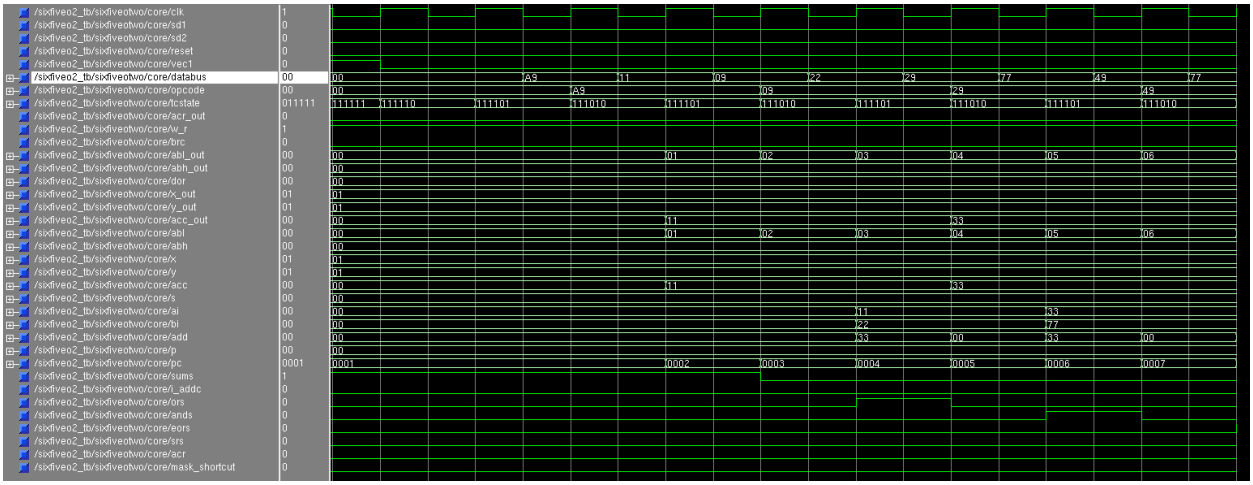
Verifying a fully VHDL-implemented CPU with over 700 possible states required careful planning of the test bench. Having tested earlier on that a digital-logic-heavy command such as one involving the ALU can stabilize an output value within one 50MHz clock cycle, the remainder of our verification effort focused on tracking all necessary transitions occurring at each rising edge of the clock.

The well-organized mask implementation of the decode ROM meant that if there was an error, it would occur across all opcodes in the same category. This also meant that the removal of the error would solve the problem for all opcodes with similar behavior. This encouraged us to approach the debug of the i6502 on a mask-by-mask basis, while keeping an eye on noticeable patterns.

2. Unit tests: Modelsim

Our first series of tests were executed on Modelsim. Modelsim allows for a combination of both synthesizable and non-synthesizable elements. This expedites the debug process by replacing non-synthesizable components with appropriate behavioral code, and also eliminating memory declaration which consumed the majority of compilation time. By feeding the databus directly with chosen opcodes and operands at each clock cycle(40ns), we were able to quickly run through an ideal-case simulation of the edge-triggered ISA behavior. Each test targeted an individual mask among: 62 opcode masks represented by aaaxxxxx, 11 address modes (xxxbbbxx), four categories (xxxxxxcc), and exceptions that require a combination of two or more of aaa, bbc, and cc. Below is an example test bench, testing LDA imm (11) and ORA imm (22), and following (TODO: figure) is the simulation result. A portion of the used testbench can be found in the appendix (TODO: point to location).

```
process  --test the first section cc=01
begin
wait for 400ns;
databus<=x"A9"; --LDA imm
wait for 40ns;
databus<= x"11";  --(LDA) #=11
wait for 40ns;
databus<= x"09"; --ORA #
wait for 40ns;
databus<= x"22"; --#=22
wait for 40ns;
 ---
end process;
```



The large database of timing diagrams on paper served their purpose well in our visual verification of the opcode action. First, we verified the large framework of each instruction, to guarantee the loading of the next instruction. This included a check on 1) decision of Predecode on total cycle number, 2) Mealy machine increment and interaction with RCL, 3) PC increment, and 4) output of instruction. The second and more detailed check verified details in 1) flag output, 2) ALU output, 3) on-and-off of ALU input signals, etc. In such a way we tested a combination of masks that comprised of about 50% of the 152 instructions, with much confidence in the functionality of the remaining instructions due to the repetitive mask structure of the RCL. These set of simulations yet were not able to test possible sources of error that the behavioral code had replaced, such as signal delay in communication with the memory, and data-fetch/write processes.

3. Hardware verification

To verify a hardware implementation after synthesis, we used both peripherals (HEX displays, LEDs) for display of critical numbers, and the SignalTap II Logic Analyzer for additional internal value checking. In detail, we displayed A, X and Y registers on hex displays, processor status register flags as green LEDs. In addition, we implemented a 2's power frequency divider to slow down the 50MHz clock to ~1Hz for real-time verification. The 6502 does not have an instruction to stop all activity, because the processor always needs to be responsive to a certain signal. The 'BRK' instruction which is closest in function resets PC to an absolute address stored in the end of the memory, and continues to function. We assigned an 'illegal' opcode "FF" the 'stop all' function, for its great utility in debugging.

Switch 17 (SW17) on the DE2 board was designated to be the reset switch. At reset, ROM.vhd writes out its initialization bits to the SRAM at every clock cycle, as well as resetting all internal register values and signals to default. ROM.vhd is declared as an internal memory array and hence consumes a considerable amount of time to compile. Initializing only the first page of the memory (0000-00FF, 156 bytes) proved sufficient to debug all opcodes with reasonable compilation time (2 minutes). Below is an example of ROM.vhd testing debug.

```
type rom_type is array (0 to 255) of std_logic_vector(7 downto 0);
constant ROM : rom_type :=
(          x"A5", x"11", x"00", x"ae", x"12", x"00", x"a0", x"aa", x"a0", x"bb", x"a2", x"cc", x"ff", x"ff", x"00", x"00",
 x"00", x"66", x"33", x"44", x"55", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
 x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
 x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
 … )
```

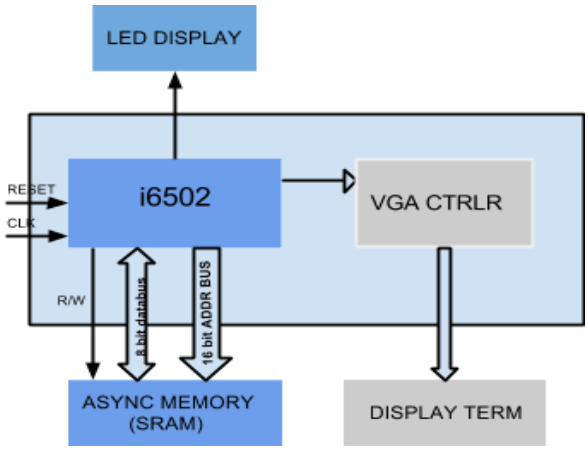
All opcodes in cc=10, 01, 00 have been verified with the following method. This sums up to about 90% of the 152 instructions (opcode + address mode).

4. Known bugs/issues

Only minor errors were found in the debug process, at a clock speed of 50MHz. This owes to the reliability of the FPGA board, small time delay in digital signal paths, and each of the team members' familiarity with timing diagrams—from which the vhdl code was implemented from. Among the few errors were: faulty PC increment, faulty reset of SUMS or I_ADDC signals, and interrupt opcodes (BRK, JSR, etc.).

V. Application

As a final product we implemented the bouncing ball as did in Lab3, but the C code replaced with the i6502 processor, as the following figure. The i6502 used the Altera DE2 video RAM for executing the bouncing ball program. We incremented the pixel index by 3 for each inc/decrement in x/y and fed the VGA raster. The VGA raster drew the circle with X and Y co-ordinates as the center.



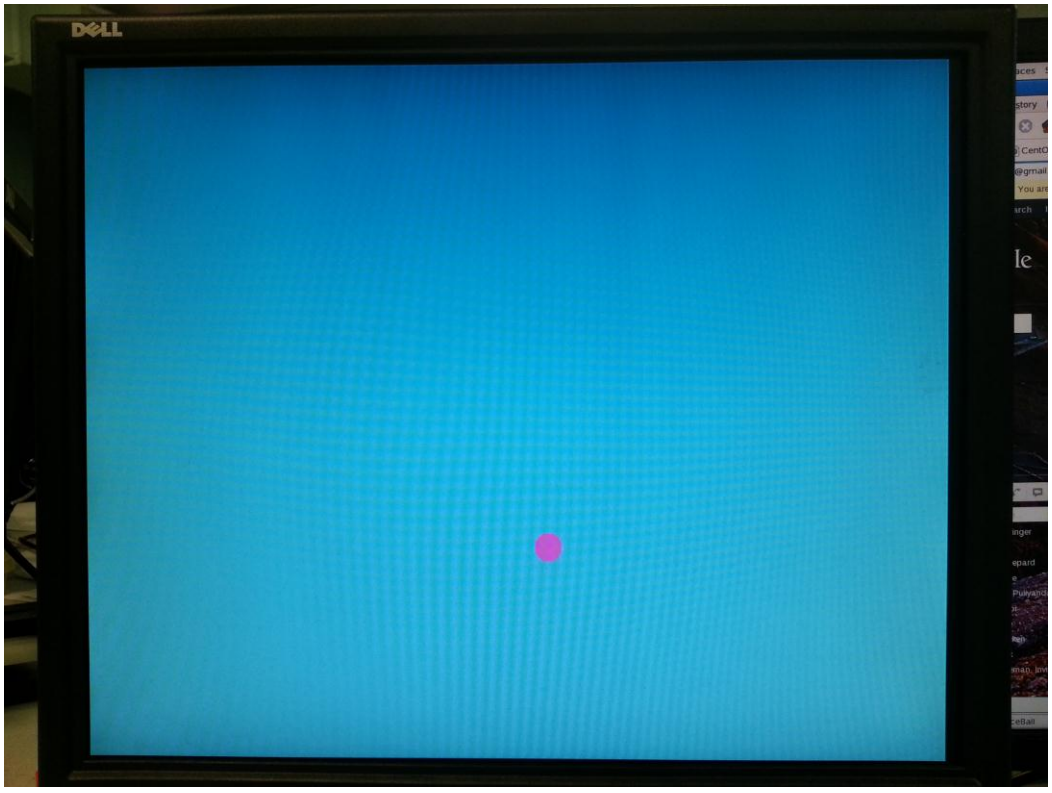
Here is a list of memory bytes used for the program, and a portion of the assembly code:

X register – x coordinate of ball
Y register – y coordinate of ball
$0070 – size of x grid
$0071 – size of y grid
$0072 – x direction of ball
$0073 – y direction of ball

--BRC1

LDA $0072	AD
CMP #$01	C9
BEQ (BRC3)	F0
BNE (BRC4)	D0

Two cross-linked loops, one for X operation and another for Y, were written in assembly code, totaling to a number of 50 instructions in twelve separate branch operations. A series of 98 bytes representing the instructions were bootloaded via ROM.vhd into the first page of the SRAM. The full code is attached in the appendix. With the reset signal off, the bouncing ball successfully showed up on the LCD screen, proving that all instructions used have been properly implemented on FPGA.



VI. Conclusion

Lessons Learned

Arthy

This project 6502 started as my most favorite project as I had taken computer arch, microarchitecture and parallel architecture. I was all excited about it. I expected this to be an experience in itself.

“Start early, develop iteratively, test in “hardware” iteratively, have small goals and accomplish them - do not sleep over the project till the last minute “

My honest advice would be as most others in previous years have reiterated - choose your team carefully.
There is nothing more important than that. Your entire experience of this class would only depend on this one factor as the entire second half of the semester is with them.

1. If you are not clear about the project when submitting the first milestone something is wrong.
2. Choose your team leader
3. Every problem you face in the team, try to solve it as team. If the team cannot solve small issues then it cannot solve the project period.
4. Choose a project which every team member is happy about. It may be hard to come to consensus but that’s team.

Anthony

It may seem rather trite to conclude a final project paper by talking about the lessons we learned, challenges we faced, and so on. Yet as we reflect back over the development of this project, there have been few, if any, projects in our undergraduate or graduate careers that have been as challenging, and consequently, rewarding as this one. Of course, this reward came with great effort. Much like hacking through a jungle with a dinner spoon, the endeavor to understand the 6502 was met with many false starts, dead ends, and futile disagreements over trivial issues like variable notation or source code control. In the end our greatest challenges arose from consequences we did not even conceive to begin with. Looking back now, it is easy to appreciate the 6502’s rich yet beautiful microarchitecture, yet, understanding and adapting its design to meet spec was a major undertaking. However, the true enemy of progress was the equally complex and foreign problem of managing communication and collaboration across a team with different interests, backgrounds, skillsets, and schedules.

Concrete progress over time was also very nonlinear. Seeing the processor come to life at quite literally last minute justified all of the sleepless nights and lost weekends. It was the pearl in the oyster bridging the gap between software and silicon is something that even most software and circuits engineers take for granted. Being able to understand the software/hardware interface from C

to instruction set down to the silicon is more rewarding than I thought it would be when we started this project. The famous computer scientist Donald Knuth once said:

"The psychological profiling [of a programmer] is mostly the ability to shift levels of abstraction, from low level to high level. To see something in the small and to see something in the large." – Donald Knuth

If I had to nail it down to a few points:

1. **Beware of "analysis paralysis":** Of course it's important to fully understand the architecture you're building but in the end what matters is execution. It's easy always easy to justify drawing more timing diagrams or understanding but this should never come at the expense of real progress.
2. **Know thy build environment:** Hours can be wasted when you spend time designing around technology that isn't fully understood.
3. **RTFM,** even if it's 300 pages of roughly scanned typewriter print.

Many of our problems came down to the fact that we had very little upfront knowledge, if any, about the architecture of the 6502. Of course hindsight is 20/20 and in retrospect it would have made more sense to design our i6502 implementation from the top-down beginning with an understanding of what the opcodes do and how many cycles they require under various conditions. In other words, the instruction set and the timing states really define the "soul" of the 6502 and do not require any assumptions about the details of the microarchitecture implementation. The controller (also called random control logic) is by far the most complex aspect of the 6502, however, its state can be defined by the current instruction and the timing state (T0-T7). From there, writing up timing diagrams plays an important role in mapping the control logic to the control lines which dictate when certain components and bus lines of the processor are asserted.

Jaebin Choi

I initially started working on the project with not much interest in VHDL coding, and little knowledge of the 6502. I took the class and VHDL from a very practical perspective, thinking that being able to communicate with hardware would be a strong skill set in my future research. I believe my relaxed attitude partly relied on the fact that I trusted the team in its ambition and intelligence. But the 6502 changed that. I felt a nerdy attachment for the intricacies of the primitive processor, and I was glad that I had, by my teammates, been exposed to such great work. However optimistic I am about my learning experience, I must admit that there has been a great cacophony inside the team. I took a part in every implementation step of the i6502, from timing diagrams to the bouncing ball. In detail, I have written a third of the decode rom (cc=00), integrated the CPU.vhd divided into three people's work, verified on both software and hardware, and written the code regarding the bouncing ball.

Yu Chen

My contribution and lesson learnt

To be honest, I contributed the most in this project. Before the third milestone, I am one of the members who spent a huge amount of time in learning the processor, reading materials, drawing timing diagrams and helping the other members to understand the system. I am also the person who had done two unsuccessful tries in implementing a single opcode in VHDL and raised the issue of different timing between latched based system and DFF based system. After the third milestone and finalizing our design strategy, I implemented and tested more than half of all the opcodes (cc=10 and 01 parts) and all the others blocks: timing generator, predecoder, DFF, ROM, SRAM controller and gave the life to the processor on FPGA. I am also one of the active members who built applications like bouncing ball and hex displacement in the final stage of the project.

Lesson learnt:

1. Looking for good team members. Good team members should not only have skills, abilities to finish tasks, passions on the project but also should be responsible reliable.

2. Not hesitate to communicate with your team members. Not ignore any new ideas, different opinions and arguments.

3. Don't try to finish things alone. You are not a superman!

Contribution Summary

Arthy: Understand and research on 6502 operation; Implemented single byte instructions in final vhdl code.

Anthony: Conducted initial research on 6502 material; Created database of opcodes and possible timing state; Created custom block diagram for the data path.

Yu: Understand and research on 6502 operation; Implemented initial vhdl test code and all of the other surrounging building blocks in vhdl for final implementation; Implemented cc=01 and cc=10 part of opcodes in final vhdl code; Test and debug; Realized Bouncing ball top level architecture and VGA, SRAM controls.

Jaebin: Implemented cc=00 of decode in final vhdl code; Integration of full decode; Test and debug; Design the bouncing ball assembly code and implementation

Future Direction:

With the fundamental design of the 6502 complete the fruits of our labor can now be realized. Our basic bouncing ball serves to demonstrate a very basic, but functional "hello, world" example of our i6502 processor. It would be worthwhile to examine and test existing ROMs on our architecture to validate our design.

In the broader picture, there exists significant interest in the 6502 processor in the "hacker" community most notably through visual6502.org and 6502.org organizations, both of which were extremely influential in our own understanding of the 6502. This project would not have been possible without their efforts in reconstructing. There is also work that we have done that has not yet been documented or published online by the visual 6502 team. We hope that we can give back to the community through some of the work that we have don

References:

1. The Visual 6502 Project
2. Hanson's 6502 Block Diagram
3. 6502 Datapath http://visual6502.org/wiki/index.php?title=6502_datapath
4. 6502.org
5. MOS Hardware Manual
6. MOS Software Manual
7. Illegal Opcodes
8. Opcode format
9. Reverse Engineering the MOS 6502 CPU
10. Beregnyei Balazs' full transistor-level schematic
11. An Interview with Donald Knuth *http://webcache.googleusercontent.com/search?q=cache:gUSdnZ4m0qkJ:www.drdobbs.com/an-interview-with-donald-knuth/184409858+&cd=4&hl=en&ct=clnk&gl=us*

Timing Diagrams used for mask generation for cc=00 (as examples).

BIT	**absolute**	**2C**	**4 cycles**	**NVZ**			
	T0	T1	T2	T3	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	1003(HH)	HHLL(AA)	1004(next)	1005
DATA		opcode	LL	HH	AA	next	
PC	1001	1002	1003	1004	1004	1005	1006
IR			opcode	opcode	opcode	opcode	next
AI				0		BB	
BI				LL		AA	
ADD				LL		result	
AC	---	---	BB	BB	BB	BB	BB
N							bit7 of AA
V							bit6 of AA
Z							1 if result=00
			PC>AB		PC>AB	PC>AB	
			PC+1>PC		PC+1>PC	PC+1>PC	
			DATA>BI	DATA>ABH	DATA>BI	DATA>IR	
			00>AI	ADD>ABL	AC>AI		
			SUMS				
					ANDS	ADD>N	
						ADD>V	
						ADD>Z	

JMP	absolute	4C	3 cycles	no flags		
	T0	T1	T2	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	1003(HH)	HHLL(next)	HH(LL+1)
DATA		opcode	LL	HH	next	
PC	1001	1002	1003	10LL	HH(LL+1)	HH(LL+2)
IR			opcode	opcode	opcode	next
AI						
BI						
ADD						
			PC>AB	[DATA,PCL]>AB	PC>AB	
			DATA>PCL	[DATA,PCL]+1>PC	PC+1>PC	

JMP	indirect	6C	5 cycles	no flags				
	T0	T1	T2	T3	T4	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	1003(HH)	HHLL(AA)	HHLL+1(BB)	BBAA(next)	BBAA+1
DATA		opcode	LL	HH	AA	BB	next	
PC	1001	1002	1003	10LL	HHLL+1	HHAA	BBAA+1	BBAA+2
IR			opcode	opcode	opcode	opcode	opcode	next
AI								
BI								
ADD								
			PC>AB	[DATA,PCL]>AB	PC>AB	[DATA,PCL]>AB	PC>AB	
			DATA>PCL	[DATA,PCL]+1>PC	DATA>PCL	[DATA,PCL]+1>PC	PC+1>PC	

STY	absolute	8C	4 cycles	no flags			
	T0	T1	T2	T3	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	1003(HH)	HHLL	1004(next)	1005
DATA		opcode	LL	HH	AA	next	
PC	1001	1002	1003	1004	1004	1005	1006
IR			opcode	opcode	opcode	opcode	next
AI				0			
BI				LL			
ADD				LL			
Y	----	----	AA	AA	AA	AA	
DOR					AA		
			PC>AB		PC>AB	PC>AB	
			PC+1>PC		PC+1>PC	PC+1>PC	
			DATA>BI	DATA>ABH	DOR>DATA	DATA>IR	
			00>AI	ADD>ABL	R/W>W		
			SUMS	Y>DOR			

LDY	absolute	AC	4 cycles	NZ			
	T0	T1	T2	T3	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	1003(HH)	HHLL(AA)	1004(next)	1005
DATA		opcode	LL	HH	AA	next	
PC	1001	1002	1003	1004	1004	1005	1006
IR			opcode	opcode	opcode	opcode	next
AI				0		0	
BI				LL		AA	
ADD				LL		AA	
Y	----	----	----	----	----	AA	AA
N							bit7 of AA
Z							1 if AA=00

			PC>AB		PC>AB	PC>AB	
			PC+1>PC		PC+1>PC	PC+1>PC	
			DATA>BI	DATA>ABH	DATA>Y	DATA>IR	
			00>AI	ADD>ABL			
			SUMS				

CPX	absolute	EC	4 cycles	NZC			
	T0	T1	T2	T3	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	1003(HH)	HHLL(AA)	1004(next)	1005
DATA		opcode	LL	HH	AA	next	
PC	1001	1002	1003	1004	1004	1005	1006
IR			opcode	opcode	opcode	opcode	next
AI				0		BB	
BI				LL		invert(AA)	
ADD				LL		result	
X	---	----	BB	BB	BB	BB	BB
N							bit7 of result(ADD)
Z							1 if X=memory
C							1 if X>=memory
			PC>AB		PC>AB	PC>AB	
			PC+1>PC		PC+1>PC	PC+1>PC	
			DATA>BI	DATA>ABH	DATAbar>BI	DATA>IR	
			00>AI	ADD>ABL	X>AI		
			SUMS				
					SUMS	ADD>N	
						ADD>Z	
						ADD>C	

BIT	zeropage	24	3 cycles	NVZ		
	T0	T1	T2	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	00LL(AA)	1003(next)	1004
DATA		opcode	LL	AA	next	
PC	1001	1002	1003	1003	1004	1005
IR			opcode	opcode	opcode	next

AI					BB	
BI					AA	
ADD					result	
AC	---	---	BB	BB	BB	BB
N						bit7 of AA
V						bit6 of AA
Z						1 if result=00
				PC>AB	PC>AB	
				PC+1>PC	PC+1>PC	
			DATA>ABL	DATA>BI	DATA>IR	
			00>ABH	AC>AI		
				ANDS	ADD>N	
					ADD>V	
					ADD>Z	

STY	zeropage	84	3 cycles	no flags		
	T0	T1	T2	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	00LL	1003(next)	1004
DATA		opcode	LL	AA	next	
PC	1001	1002	1003	1003	1004	1005
IR			opcode	opcode	opcode	next
AI						
BI						
ADD						
Y	---	---	AA	AA	AA	AA
DOR				AA	AA	AA
				PC>AB	PC>AB	
				PC+1>PC	PC+1>PC	
			DATA>ABL	DOR>DATA	DATA>IR	

			00>ABH	R/W>W		
			Y>DOR			

LDY	zeropage	A4	3 cycles	NZ		
	T0	T1	T2	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	00LL(AA)	1003(next)	1004
DATA		opcode	LL	AA	next	
PC	1001	1002	1003	1003	1004	1005
IR			opcode	opcode	opcode	next
AI					0	
BI					AA	
ADD					AA	
Y	---	---	---	---	AA	AA
N						bit7 of AA
Z						1 if AA=00
				PC>AB	PC>AB	
				PC+1>PC	PC+1>PC	
			DATA>ABL	DATA>Y	DATA>IR	
			00>ABH			
				*LDY affects flags N,Z. Does the memory value have to go through ADD?		
				DATA>BI	ADD>N	
				00>AI	ADD>Z	
				SUMS		

CPX	zeropage	E4	3 cycles	NZC		
	T0	T1	T2	T0	T1	T2

ADDR	1000	1001(op)	1002(LL)	00LL(AA)	1003(next)	1004
DATA		opcode	LL	AA	next	
PC	1001	1002	1003	1003	1004	1005
IR			opcode	opcode	opcode	next
AI					BB	
BI					invert(AA)	
ADD					result	
X	---	---	BB	BB	BB	BB
N						bit7 of result(ADD)
Z						1 if X=memory
C						1 if X>=memory
				PC>AB	PC>AB	
				PC+1>PC	PC+1>PC	
			DATA>ABL	DATAbar>BI	DATA>IR	
			00>ABH	X>AI		
				SUMS	ADD>N	
					ADD>Z	
					ADD>C	

STY	zeropage,X	94	4 cycles	no flags			
	T0	T1	T2	T3	T0	T1	T2
ADDR	1000	1001(op)	1002(ZZ)	1002(ZZ)	00RR	1003(next)	1004
DATA		opcode	ZZ	ZZ	MM	next	
PC	1001	1002	1003	1003	1003	1004	1005
IR			opcode	opcode	opcode	opcode	next
AI				VV			
BI				ZZ			
ADD				RR			
X	---	---	VV	VV	VV	VV	VV
Y	---	---	MM	MM	MM	MM	MM
DOR					MM		

					PC>AB	PC>AB	
					PC+1>PC	PC+1>PC	
			DATA>BI		DOR>DATA	DATA>IR	
			X>AI	ADD>ABL	R/W>W		
			SUMS	00>ABH			
				Y>DOR			

LDY	zeropage,X	B4	4 cycles	NZ			
	T0	T1	T2	T3	T0	T1	T2
ADDR	1000	1001(op)	1002(ZZ)	1002(ZZ)	00RR(MM)	1003(next)	1004
DATA		opcode	ZZ	ZZ	MM	next	
PC	1001	1002	1003	1003	1003	1004	1005
IR			opcode	opcode	opcode	opcode	next
AI				VV		0	
BI				ZZ		MM	
ADD				RR		MM	
X	---	---	VV	VV	VV	VV	VV
Y	---	---	---	---	---	MM	MM
N							bit7 of MM
Z							1 if MM=00
					PC>AB	PC>AB	
					PC+1>PC	PC+1>PC	
			DATA>BI		DATA>Y	DATA>IR	
			X>AI	ADD>ABL			
			SUMS	00>ABH			
					DATA>BI	ADD>N	
					00>AI	ADD>Z	
					SUMS		

LDY	absolute,X	BC	4-5 cycles	NZ			
	T0	T1	T2	T3	T0	T1	T2
ADDR	1000	1001(op)	1002(LL)	1003(HH)	HHSS(AA)	1004(next)	1005

DATA		opcode	LL	HH	AA	next	
PC	1001	1002	1003	1004	1004	1005	1006
IR			opcode	opcode	opcode	opcode	next
AI				XX	0	0	
BI				LL	HH	AA	
ADD				SS	HH	AA	
X	---	---	XX	XX	XX	XX	XX
Y	---	---	---	---	--	AA	AA
N							bit7 of AA
Z							1 if AA=00
alucout				0			
			PC>AB		PC>AB	PC>AB	
			PC+1>PC		PC+1>PC	PC+1>PC	
			DATA>BI	DATA>ABH	DATA>Y	DATA>IR	
			X>AI	ADD>ABL			
			SUMS				
				alucout>alucin			
				DATA>BI	DATA>BI	ADD>N	
				00>AI	00>AI	ADD>Z	
				SUMS	SUMS		
				if alucout=0			
				nothing happens			

LDY	immediate	A0	2 cycles	NZ	
	T0	T1	T2+T0	T1	T2
ADDR	1000	1001(op)	1002(VV)	1003(next)	1004
DATA		opcode	VV	next	
PC	1001	1002	1003	1004	1005
IR			opcode	opcode	next
AI				0	
BI				VV	
ADD				VV	
Y	---	---	---	VV	VV
N					bit7 of

					VV
Z					1 if VV=00
			PC>AB	PC>AB	
			PC+1>PC	PC+1>PC	
			DATA>Y	DATA>IR	
			DATA>BI	ADD>N	
			00>AI	ADD>Z	
			SUMS		

CPY	immediate	C0	2 cycles	NZC	
	T0	T1	T2+T0	T1	T2
ADDR	1000	1001(op)	1002(VV)	1003(next)	1004
DATA		opcode	VV	next	
PC	1001	1002	1003	1004	1005
IR			opcode	opcode	next
AI				AA	
BI				invert(VV)	
ADD				result	
Y	---	---	AA	AA	AA
N					bit7 of result(ADD)
Z					1 if Y=memory
C					1 if Y>=memory
			PC>AB	PC>AB	
			PC+1>PC	PC+1>PC	
			DATAbar>BI	DATA>IR	
			Y>AI		
			SUMS		
				ADD>N	
				ADD>Z	
				ADD>C	

CODES

DE2_VGA_RASTER.VHD

```
--------------------------------------------------------------------------------
--
-- Simple VGA raster display
--
-- Stephen A. Edwards
-- sedwards@cs.columbia.edu
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is

  port (
    reset : in std_logic;
    clk   : in std_logic;                    -- Should be 25.125 MHz

    center: in std_logic_vector(15 downto 0) := X"f0f0"; -- circle center
    --chipselect : in std_logic;
    --write      : in std_logic;
    --address  :  in std_logic_vector(17 downto 0);
    --readdata : out std_logic_vector(15 downto 0);
    --writedata : in std_logic_vector(15 downto 0);

    VGA_CLK,                            -- Clock
    VGA_HS,                             -- H_SYNC
    VGA_VS,                             -- V_SYNC
    VGA_BLANK,                          -- BLANK
    VGA_SYNC : out std_logic;           -- SYNC
    VGA_R,                              -- Red[9:0]
    VGA_G,                              -- Green[9:0]
    VGA_B : out unsigned(9 downto 0) -- Blue[9:0]
    );

end de2_vga_raster;

architecture rtl of de2_vga_raster is

  -- Video parameters
```

```
  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  --constant RECTANGLE_HSTART : integer := 100;
  --constant RECTANGLE_HEND   : integer := 240;
  --constant RECTANGLE_VSTART : integer := 100;
  --constant RECTANGLE_VEND   : integer := 180;

  -- Signals related to ball drawing
  constant RADIUS       : integer :=10;   --radius of the ball
  constant Hinitial : integer :=400;  --initial x value of the center of the
ball
  constant Vinitial : integer :=263;  --initial y value of the center of the
ball


  -- Signals for the video controller
  signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
  signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;

  signal vga_hblank, vga_hsync,
    vga_vblank, vga_vsync : std_logic;  -- Sync. signals

  signal rectangle_h, rectangle_v, rectangle : std_logic;  -- rectangle area

 -- signal center_in : unsigned(31 downto 0) := X"008000c0";

begin

  -- Horizontal and vertical counters

  HCounter : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        Hcount <= (others => '0');
      elsif EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
```

```
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      Vcount <= (others => '0');
    elsif EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' or EndOfLine = '1' then
      vga_hsync <= '1';
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_hblank <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      vga_hblank <= '1';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
```

```
        vga_vsync <= '1';
      elsif EndOfLine ='1' then
        if EndOfField = '1' then
          vga_vsync <= '1';
        elsif Vcount = VSYNC - 1 then
          vga_vsync <= '0';
        end if;
      end if;
    end if;
  end process VSyncGen;

  VBlankGen : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        vga_vblank <= '1';
      elsif EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;


BallGen : process (clk)
variable distance_square : integer;
variable distance_H      : integer;
variable distance_V      : integer;
begin

      if rising_edge (clk) then
      distance_H := abs(TO_INTEGER(Hcount)- 3*TO_INTEGER(unsigned(center(7
downto 0)))-144);
      distance_V := abs(TO_INTEGER(Vcount)- 3*TO_INTEGER(unsigned(center(15
downto 8)))-35);
--    distance_H := abs(TO_INTEGER(Hcount)- Hinitial);
--    distance_V := abs(TO_INTEGER(Vcount)- Vinitial);
      distance_square :=(distance_H*distance_H)+(distance_V*distance_V);
            if reset = '1' then
            rectangle_h<= '0';
            rectangle_v<= '0';
            elsif distance_square < RADIUS*RADIUS then
            rectangle_h<= '1';
            rectangle_v<= '1';
            else
            rectangle_h<= '0';
            rectangle_v<= '0';
            end if;
      end if;
```

```
end process BallGen;


  rectangle <= rectangle_h and rectangle_v;

  -- Registered video signals going to the video DAC

  VideoOut: process (clk, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk'event and clk = '1' then
      if rectangle = '1' then --color of ball
        VGA_R <= "1111111111";
        VGA_G <= "0000000000";
        VGA_B <= "1111111111";
      elsif vga_hblank = '0' and vga_vblank ='0' then
        VGA_R <= "0111011100"; --color of background
        VGA_G <= "1110000000";
        VGA_B <= "1110111000";
      else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;

  VGA_CLK <= clk;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```

DEBOUNCE.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--the delay settings has been changed to enable efficient simulations.
--original settings for the board: 24 bits for 'count'
--new settings for simulation: 7 bits for 'count'
```

```
entity debounce is
      port (
            clk, resetsw : in std_logic;
            resetout  : out std_logic
            );
end debounce;

architecture imp of debounce is
signal count: unsigned(23 downto 0) := (others => '0');
signal rout_buf: std_logic := '0';
begin

      detect: process(clk)
      begin

      if rising_edge(clk) then

            --0 to 1 transition of resetsw
            if rout_buf='0' and resetsw='1' then
                  count <= count + 1;
                  if count(23)='1' then
                        rout_buf <= '1';
                        resetout <= '1';
                        count <= (others => '0'); -- reset count.
                  end if;
            end if;

            --1 to 0 transition of resetsw
            if rout_buf='1' and resetsw='0' then
                  count <= count + 1;
                  if count(23)='1' then
                        rout_buf <= '0';
                        resetout <= '0';
                        count <= (others => '0'); -- reset count.
                  end if;
            end if;

      end if;

      end process detect;

end imp;
```

HEX7SEG.VHD

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;
-- Provides the unsigned type

entity hex7seg is
      port
            ( input : in std_logic_vector(3 downto 0); -- A number
        output : out std_logic_vector(6 downto 0)); -- Just bits
end hex7seg;

architecture combinational of hex7seg is
begin
      with input select output <=
            "1000000" when x"0",
            "1111001" when x"1",
            "0100100" when x"2",
            "0110000" when x"3",
            "0011001" when x"4",
            "0010010" when x"5",
            "0000010" when x"6",
            "1111000" when x"7",
            "0000000" when x"8",
            "0010000" when x"9",
            "0001000" when x"A",
            "0000011" when x"B",
            "1000110" when x"C",
            "0100001" when x"D",
            "0000110" when x"E",
            "0001110" when x"F",
            "XXXXXXX" when others
;
end combinational;
```

PREDECODE.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Predecode is
port (
            databus   : in     std_logic_vector(7 downto 0);
            reset     : in  std_logic;
            cycle_number : out    unsigned(3 downto 0);
            Instruction  : out    std_logic_vector(7 downto 0);
            RMW    : out std_logic);
end Predecode;
```

```
architecture rtl of Predecode is

begin
      process(databus, reset)
      begin
      if reset='1' then
            cycle_number <= x"1";  --yuchen0513
            Instruction <= x"00";
            RMW <= '0';

      else
            Instruction <= databus;
            RMW <= '0';

            if databus=x"FF" then
                  cycle_number <= x"1";

      --================================cc=01
section==================================
            elsif databus(1 downto 0)="01" then
                  if databus(4 downto 0)= "00001" then --(zero page, X) with
cc=01
                        cycle_number <= x"6";
                        RMW <= '0';
                  elsif databus(4 downto 0) = "00101" then --zero page with
cc=01
                        cycle_number <= x"3";
                        RMW <= '0';
                  elsif databus(4 downto 0) = "01001" then --#immediate with
cc=01
                        cycle_number <= x"2";
                        RMW <= '0';
                  elsif databus(4 downto 0) = "01101" then --absolute with
cc=01
                        cycle_number <= x"4";
                        RMW <= '0';
                  elsif databus(4 downto 0) = "10001" then --(zero page), Y
with cc=01
                        cycle_number <= x"6";
                        RMW <= '0';
                  elsif databus(4 downto 0) = "10101" then --zero page, X
with cc=01
                        cycle_number <= x"4";
                        RMW <= '0';
                  elsif databus(4 downto 0) = "11001" then --absolute, Y with
cc=01
                        cycle_number <= x"5";
                        RMW <= '0';
                  elsif databus(4 downto 0) = "11101" then --absolute, X with
cc=01
                        cycle_number <= x"5";
                        RMW <= '0';
```

```
                  else
                        cycle_number<=x"1"; --JB0513
                        RMW <= '0';
                  end if;
      --================================cc=01 section
ends==============================


      --================================cc=10
section==================================
            elsif databus(1 downto 0)="10" then
                  --Arthy's code, hex XA: 1xxx1010
                  if databus(7)='1' and databus(3 downto 2)="10" then
                        RMW <= '0';
                        cycle_number<= x"2"; --JB0511. check if wrong.

                  --Yu's code below..
                  --STX, LDX (non read-modify-write code)
                  elsif databus(7 downto 6)="10" and not(databus(3 downto
2)="10") then --************arthy
                        RMW <= '0';
                        if databus(4 downto 2)="000" then --immediate
                              cycle_number<= x"2";
                        elsif databus(4 downto 2)="001" then --zero page
                              cycle_number<= x"3";
                        elsif databus(4 downto 2)="010" then --accumulator
****************arthy
                              cycle_number<= x"2";
                        elsif databus(4 downto 2)="011" then --absolute
                              cycle_number<= x"4";
                        elsif databus(4 downto 2)="101" then --zero page, X/Y
                              cycle_number<= x"4";
                        elsif databus(4 downto 2)="111" then  --absolute, X/Y
                              cycle_number<= x"5";
                        else cycle_number<=x"0"; RMW <= '0';
                        end if;
                  --6 read-modify-write instructions
                  elsif databus(4 downto 2)="010" then --accumulator
                        cycle_number<= x"2";
                        RMW <= '0';
                  else
                        RMW <= '1';
                        if databus(4 downto 2)="001" then cycle_number<=x"5";
--zero page
                        elsif databus(4 downto 2)="011" then
cycle_number<=x"6"; --absolute
                        elsif databus(4 downto 2)="101" then
cycle_number<=x"6"; --zero page, X/Y
                        elsif databus(4 downto 2)="111" then
cycle_number<=x"7"; --absolute, X/Y
                        else cycle_number<=x"1";  --yuchen0513
                        end if;
```

```
                end if;
     --================================cc=10 section
ends==================================

       --================================cc=00
section==================================
          elsif databus(1 downto 0)="00" then
               if databus(4 downto 2)="000" and databus(7)='0' then --
interrupts
                    if databus(6 downto 5) = "00" then  --BRK
                         --cycle_number<=x"7"; --JB need to define VEC
separately!
                         cycle_number<=x"1"; --yuchen0513
                         RMW <= '0';
                    else  --JSR, RTS, RTI
                         cycle_number<=x"6";
                         RMW <= '0';
                    end if;

               else -- among cc=00, all other than interrupts

                    --Arthy's hex: X8 codes fit here.
                    if databus(3 downto 2)="10" then
                         RMW <= '0';
                         if    databus(7 downto 4)="0000" then
cycle_number<=x"3";
                         elsif databus(7 downto 4)="0010" then
cycle_number<=x"4";
                         elsif databus(7 downto 4)="0100" then
cycle_number<=x"3";
                         elsif databus(7 downto 4)="0110" then
cycle_number<=x"4";
                         else
                  cycle_number<=x"2";
                         end if;
                    --end of Arthy's X8 codes.

                    elsif databus(4 downto 2)="100" then --branch
                         cycle_number<=x"0"; -- 2 for no branch, 3 for
branch, 4 for branch w/ page crossing. JB0510: zero.
                         --BRC <= '1'; JB0510 commented out. BRC value
is determined by CPU in cycle T2.
                         RMW <= '0';
                    --else cycle_number<=x"0"; RMW <= '0';
                    --end if;
                    elsif databus(4 downto 2)="000" then --immediate
                         cycle_number<=x"2";
                         RMW <= '0';
                    --else cycle_number<=x"0"; RMW <= '0';
                    --end if;
                    elsif databus(4 downto 2)="001" then --zeropage
                         cycle_number<=x"3";
```

```
                                RMW <= '0';
                        --else cycle_number<=x"0"; RMW <= '0';
                        --end if;
                        elsif databus(4 downto 2)="011" and databus(7 downto
5)="010" then -- absolute, JMP abs
                                cycle_number<=x"3";
                                RMW <= '0';
                        elsif databus(4 downto 2)="011" and databus(7 downto
5)="011" then -- absolute, JMP ind
                                cycle_number<=x"5";
                                RMW <= '0';
                        elsif databus(4 downto 2)="011" and not(databus(7
downto 6)="01") then --rest of all absolutes
                                cycle_number<=x"4";
                                RMW <= '0';
                        --else cycle_number<=x"0"; RMW <= '0';
                        --end if;
                        elsif databus(4 downto 2)="101" then --zeropage,X
                                cycle_number<=x"4";
                                RMW <= '0';
                        --else cycle_number<=x"0"; RMW <= '0';
                        --end if;
                        elsif databus(4 downto 2)="111" then --absolute,X
                                cycle_number<=x"5"; --could be 4 w/o page
crossing
                                RMW <= '0';
                        else cycle_number<=x"1"; RMW <= '0';  --yuchen0513
                        end if;
                end if;
            else cycle_number<=x"1"; RMW <= '0';
            end if;
      --================================cc=01 section
ends==============================

      end if;
      end process;
end rtl;
```

QUASITOPLEVEL.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity QuasiTopLevel is
port(
               CLOCK_50 : std_logic;
```

```
                 HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
                          : out std_logic_vector(6 downto 0);          --
(active low)
                SW : in std_logic;          -- DPDT switches
                SRAM_DQ : inout unsigned(15 downto 0);          -- Data bus
16 Bits
                SRAM_ADDR : out unsigned(17 downto 0);          -- Address
bus 18 Bits
                SRAM_UB_N,                                      -- High-byte
Data Mask
                SRAM_LB_N,                                      -- Low-byte
Data Mask
                SRAM_WE_N,                                      -- Write
Enable
                SRAM_CE_N,                                      -- Chip
Enable
                SRAM_OE_N : out std_logic                       -- Output
Enable

);
end QuasiTopLevel;

architecture datapath of QuasiTopLevel is
signal Databus, DOR, ROM_data : std_logic_vector(7 downto 0);
signal Addrbus, ROM_address  : std_logic_vector(15 downto 0);
signal W_R : std_logic;

component SixFiveO2
port(
           Databus :in std_logic_vector(7 downto 0);
           Addrbus :out std_logic_vector(15 downto 0);
           DOR     : out std_logic_vector(7 downto 0);
      reset, clk   :in std_logic;
           XL, XH, YL, YH, ACCL, ACCH : out std_logic_vector(6 downto 0);
           W_R     : out std_logic);
end component;

component rom is
port(addr  : in std_logic_vector(15 downto 0);
           data : out std_logic_vector(7 downto 0));
end component;

component SRAMCtrl is
port (
           reset, clk, W_R : in  std_logic;
           ROM_data, DOR  : in std_logic_vector(7 downto 0);
           databus    : out std_logic_vector(7 downto 0);
           AddressBus : in std_logic_vector(15 downto 0);
           ROM_address : out std_logic_vector(15 downto 0);
           SRAM_DQ    : inout unsigned(15 downto 0);
           SRAM_ADDR  : out unsigned(17 downto 0);
           SRAM_UB_N,            -- High-byte Data Mask
```

```
            SRAM_LB_N,              -- Low-byte Data Mask
            SRAM_WE_N,              -- Write Enable
            SRAM_CE_N,              -- Chip Enable
            SRAM_OE_N : out std_logic  -- Output Enable
);
end component;

begin

CPUConnect: SixFiveO2 port map(clk=>CLOCK_50, reset=>SW, W_R=>W_R, XH=>HEX7,
XL=>HEX6, YH=>HEX5, YL=>HEX4, ACCH=>HEX3, ACCL=>HEX2,
                                                        Databus=>Databus,
DOR=>DOR, Addrbus=>Addrbus);

InstructionROM: Rom port map(addr=>ROM_address, data=>ROM_data);

MemorySRAM: SRAMCtrl port map(reset=>SW, clk=>CLOCK_50, W_R=>W_R,
ROM_data=>ROM_data, DOR=>DOR,
                                                      databus=>databus,
AddressBus=>Addrbus, ROM_address=>ROM_address,
                                                      SRAM_DQ=>SRAM_DQ,
                                                      SRAM_ADDR=>SRAM_ADDR,
                                                      SRAM_UB_N=>SRAM_UB_N,
                                                      SRAM_LB_N=>SRAM_LB_N,
                                                      SRAM_WE_N=>SRAM_WE_N,
                                                      SRAM_CE_N=>SRAM_CE_N,
                                                     SRAM_OE_N=>SRAM_OE_N);
end datapath;
```

ROM.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom is
port(addr  : in std_logic_vector(15 downto 0);
           data : out std_logic_vector(7 downto 0));
end rom;

architecture imp of rom is
signal address : unsigned(7 downto 0);
type rom_type is array (0 to 255) of std_logic_vector(7 downto 0);
constant ROM : rom_type :=
(
-- the 6 lines below are for the LDA, LDX and LDY opcodes in all the
different address mode.
```

```
-- x"A9", x"11", x"a2", x"22", x"a0", x"33", x"a5", x"20", x"b5", x"20",
x"ad", x"21", x"00", x"bd", x"21", x"00",
--    x"B9", x"21", x"00", x"a1", x"22", x"b1", x"22", x"ff", x"ff", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",
--    x"ab", x"33", x"22", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",
--    x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",
--    x"00", x"00", x"42", x"43", x"20", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",
--    x"00", x"00", x"00", x"00", x"54", x"55", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",
--    x"00", x"00", x"00", x"00", x"00", x"00", x"66", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",

--the two lines below are for the INC, DEC opcodes test with immediate
address mode.
--    x"EE", x"20", x"00", x"AD", x"20", x"00", x"EE", x"20", x"00", x"AD",
x"20", x"00", x"EE", x"20", x"00", x"AD",
--    x"20", x"00", x"CE", x"20", x"00", x"AD", x"20", x"00", x"CE", x"20",
x"00", x"AD", x"20", x"00", x"ff", x"ff",

--the two lines below are for the CMP absolute test.
--    x"a9", x"12", x"cd", x"10", x"00", x"cd", x"11", x"00", x"cd", x"12",
x"00", x"ff", x"ff", x"00", x"00", x"00",
--    x"11", x"12", x"13", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",

--the 3 lines below are for CPX test with immediate and absolute address
modes.
--    x"E0", x"21", x"E0", x"22", x"E0", x"23", x"ff", x"ff", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",
--    x"EC", x"10", x"00", x"EC", x"11", x"00", x"EC", x"12", x"00", x"ff",
x"ff", x"00", x"00", x"00", x"00", x"00",
--    x"23", x"22", x"21", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",

--the 2 lines below are for CPY test with absolute address mode.
--    x"CC", x"10", x"00", x"CC", x"11", x"00", x"CC", x"12", x"00", x"ff",
x"ff", x"00", x"00", x"00", x"00", x"00",
--    x"33", x"32", x"31", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00",

--the line below is for INC, LDA and JMP loop, it will keep increase the
value in ACC.
--    x"ee", x"10", x"00", x"ad", x"10", x"00", x"4c", x"00", x"00", x"ff",
x"ff", x"00", x"00", x"00", x"00", x"00",

     x"A2", x"00", x"A0", x"00", x"A9", x"A0", x"8D", x"70", x"00", x"A9",
x"D6", x"8D", x"71", x"00", --Init1/2 (24, x18)
     x"A9", x"01", x"8D", x"72", x"00", x"A9", x"01", x"8D", x"73", x"00",
--Init2/2 (24, x18)
```

```
      x"AD", x"72", x"00", x"C9", x"01", x"F0", x"0B", x"D0", x"11", --B1:
0018 (9) --B3 and B4
      x"AD", x"73", x"00", x"C9", x"01", x"F0", x"11", x"D0", x"17", --B2:
0021 (9) --B5 and B6
      x"E8", x"EC", x"70", x"00",             x"F0", x"18", x"D0", x"39", --
B3: 002A (8) --B7 and B12
      x"CA", x"E0", x"00",               x"F0", x"19", x"D0", x"32", --B4:
0032 (7) --B8 and B120
      x"C8", x"CC", x"71", x"00",        x"F0", x"19", x"D0", x"23", --B5:
0039 (8) --B9 and B11
      x"88", x"C0", x"00",               x"F0", x"1A", x"D0", x"20", --B6:
0041 (7) --B10 and B110
      x"A9", x"00", x"8D", x"72", x"00", x"4C", x"21", x"00", --B7: 0048 (8)
--J2
      x"A9", x"01", x"8D", x"72", x"00", x"4C", x"21", x"00", --B8: 0050 (8)
--J2
      x"A9", x"00", x"8D", x"73", x"00", x"4C", x"18", x"00", --B9: 0058 (8)
--J1
     x"A9", x"01", x"8D", x"73", x"00", x"4C", x"18", x"00", --B10: 0060 (8)
--J1
      x"4C", x"18", x"00", --B11: 0068 (3) --J1
      x"4C", x"21", x"00", --B12: 006B (3) --J2
      x"ff", x"ff", --6
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --7
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --8
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --9
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --10
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --11
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --12
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --13
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", --14
      x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00" --15

      --B1: 0018
      --B2: 0021
      --B3: 002A
      --B4: 0032
      --B5: 0039
      --B6: 0041
      --B7: 0048
      --B8: 004F
      --B9: 0056
      --B10: 005D
```

```
      --B11: 0064
      --B12: 0066


      );

begin
      address<=unsigned(addr(7 downto 0));
      data<=ROM(TO_Integer(address));
end imp;
```

SIXFIVEO2.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity SixFiveO2 is
port(
            Databus :in std_logic_vector(7 downto 0);
            Addrbus :out std_logic_vector(15 downto 0);
            DOR , P, X_Reg_out, Y_Reg_out : out std_logic_vector(7 downto 0);
      reset, clk   :in std_logic;
            XL, XH, YL, YH, ACCL, ACCH : out std_logic_vector(6 downto 0);
            W_R      : out std_logic);
end SixFiveO2;

architecture imp of SixFiveO2 is
signal instruction, opcode : std_logic_vector(7 downto 0);
signal tcstate : std_logic_vector(5 downto 0);
signal cycle_number : unsigned(3 downto 0);
signal BRC, ACR, RMW, SYNC, SD1, SD2, VEC1: std_logic;
--signal DOR, databus  : std_logic_vector(7 downto 0);
--signal Addrbus: std_logic_vector(15 downto 0);
signal  ACC_Reg, X_Reg, Y_Reg  : std_logic_vector(7 downto 0);

component Predecode
port (
            databus  : in     std_logic_vector(7 downto 0);
            reset    : in  std_logic;
            cycle_number : out    unsigned(3 downto 0);
            Instruction  : out    std_logic_vector(7 downto 0);
            RMW    : out std_logic);
end component;
```

```
component DFlipFlop
port(
            input  : in std_logic_vector(7 downto 0);
            enable : in  std_logic;
            clk    : in std_logic;
            reset  : in std_logic;
            output : out std_logic_vector(7 downto 0));
end component;

component TG
port(
            clk         : in std_logic;
            cycle_number      : in unsigned(3 downto 0);
            RMW    : in std_logic;  --read-modify-write instruction
            ACR    : in std_logic;  --carry in from ALU
            BRC    : in std_logic;  --branch flag
            reset  : in std_logic;
            tcstate     : out     std_logic_vector(5 downto 0);
            SYNC, SD1, SD2    : out std_logic;
            VEC1   : out std_logic);
end component;

component CPU
port(
            clk, SD1, SD2, reset, VEC1    : in std_logic;
            opcode : in std_logic_vector(7 downto 0);
            tcstate : in std_logic_vector(5 downto 0);
            databus : in std_logic_vector(7 downto 0);
            ACR_out, W_R, BRC     : out std_logic;
            ABL_out, ABH_out, DOR, X_out, Y_out, ACC_out, P_out : out
std_logic_vector(7 downto 0)
            );
end component;

--component Memory
--    port (
--    clk, reset : in std_logic;
--    we : in std_logic;
--    address : in std_logic_vector(15 downto 0);
--    di : in std_logic_vector(7 downto 0);
--    do : out std_logic_vector(7 downto 0)
--   );
--end component;

component hex7seg
port
            ( input : in std_logic_vector(3 downto 0); -- A number
       output : out std_logic_vector(6 downto 0)); -- Just bits
end component;
```

```
begin
PredecodeLogic: Predecode port map(databus=>databus, reset=>reset,
cycle_number=>cycle_number, Instruction=>Instruction, RMW=>RMW);

IR: DFlipFlop port map(input=>instruction, enable=>SYNC, clk=>clk,
reset=>reset, output=>opcode);

Timing: TG port map(clk=>clk, cycle_number=>cycle_number, RMW=>RMW, ACR=>ACR,
BRC=>BRC, reset=>reset, tcstate=>tcstate, SYNC=>SYNC, SD1=>SD1, SD2=>SD2,
VEC1=>VEC1);

Core: CPU port map(BRC=>BRC, clk=>clk, SD1=>SD1, SD2=>SD2, VEC1=>VEC1,
reset=>reset, opcode=>opcode, tcstate=>tcstate, databus=>databus,
ABL_out=>Addrbus(7 downto 0),
                   ABH_out=>Addrbus(15 downto 8), DOR=>DOR, ACR_out=>ACR,
W_R=>W_R, X_out=>X_Reg, Y_out=>Y_Reg, ACC_out=>ACC_Reg, P_out=>P);

--Mem: Memory port map(clk=>clk, reset=>reset, we=>W_R, address=>Addrbus,
di=>DOR, do=>databus);

XHDis: hex7seg port map(input=>X_Reg(7 downto 4), output=>XH);
XLDis: hex7seg port map(input=>X_Reg(3 downto 0), output=>XL);

YHDis: hex7seg port map(input=>Y_Reg(7 downto 4), output=>YH);
YLDis: hex7seg port map(input=>Y_Reg(3 downto 0), output=>YL);

ACCHDis: hex7seg port map(input=>ACC_Reg(7 downto 4), output=>ACCH);
ACCLDis: hex7seg port map(input=>ACC_Reg(3 downto 0), output=>ACCL);

process(X_Reg, Y_Reg)
begin
X_Reg_out<=X_Reg;
Y_Reg_out<=Y_Reg;
end process;

end imp;
```

SLOWCLK.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--the delay settings has been changed to enable efficient simulations.
--25 bits for ~1/3sec
--26 bits for slower
```

```
entity slowclk is
      port (
            clkin : in std_logic;
            clkout  : out std_logic
            );
end slowclk;

architecture imp of slowclk is

signal cur: std_logic := '0';
signal count: unsigned(12 downto 0) := (others => '0');

begin

      detect: process(clkin)
      begin

            if rising_edge(clkin) then
                  count <= count + 1;
                  if count(12)='1' then
                        count <= (others => '0'); -- reset count.
                        cur <= not(cur);
                        clkout <= cur;
                  end if;
            end if;
      end process detect;
end imp;
```

SRAMCTRL.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity SRAMCtrl is
port (
            reset, clk, W_R : in  std_logic;
            ROM_data, DOR  : in std_logic_vector(7 downto 0);
            databus     : out std_logic_vector(7 downto 0);
            AddressBus : in std_logic_vector(15 downto 0);
            ROM_address : out std_logic_vector(15 downto 0);
            SRAM_DQ     : inout unsigned(15 downto 0);
            SRAM_ADDR  : out unsigned(17 downto 0);
```

```
            SRAM_UB_N,              -- High-byte Data Mask
            SRAM_LB_N,              -- Low-byte Data Mask
            SRAM_WE_N,              -- Write Enable
            SRAM_CE_N,              -- Chip Enable
            SRAM_OE_N : out std_logic  -- Output Enable
);
end SRAMCtrl;

architecture rtl of SRAMCtrl is
--signal address : unsigned(15 downto 0):=x"0000";
signal counter1: unsigned(3 downto 0) :=x"0";
begin
      process(reset, clk, W_R, ROM_data, DOR, AddressBus, SRAM_DQ)
            variable address : unsigned(15 downto 0):=x"0000";
            begin
                  SRAM_ADDR(17 downto 16)<="00";
                  SRAM_CE_N<='0';
                  SRAM_LB_N<='0';
                  SRAM_UB_N<='1';


                  if reset='1' then
                  SRAM_WE_N<='0';
                  SRAM_OE_N<='1';
                  databus<=(others=>'0');
                        if rising_edge(clk) then
                              if (counter1=x"0" or counter1=x"1")  then
counter1<=counter1+1;
                              else counter1<=counter1;
                              end if;
                              --to make sure that address=x"0000" is written
correctly
                              if (counter1=x"0" or counter1=x"1") then
address:=address;
                                    elsif address=x"ffff" then address:=x"0000";
                                     else  address:=address+1;
                              end if;
                        end if;

                        if address(15 downto 8)=x"00" then SRAM_DQ(7 downto
0)<=unsigned(ROM_data);
                        --elsif address(15 downto 0)=x"fffe" then SRAM_DQ(7
downto 0)<=x"fd";
                        --elsif address(15 downto 0)=x"ffff" then SRAM_DQ(7
downto 0)<=x"ff";
                        else SRAM_DQ(7 downto 0)<=x"00";
                        end if;
                  --SRAM_DQ(15 downto 8)<=x"00";
                  SRAM_ADDR(15 downto 0)<=address;
                  ROM_address<=std_logic_vector(address);
```

```
                else
                SRAM_WE_N<=W_R;
                SRAM_OE_N<=not (W_R);
                SRAM_ADDR(15 downto 0)<=unsigned(AddressBus);
                      if W_R='0' then
                            SRAM_DQ(7 downto 0)<=unsigned(DOR);
                            databus<=(others=>'0');
                      else SRAM_DQ(7 downto 0)<=(others=>'Z');
                            databus<=std_logic_vector(SRAM_DQ(7 downto 0));
                      end if;
                ROM_address<=(others=>'0');

                end if;
          end process;
end rtl;
```

TG.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity TG is

      port(
            clk         : in std_logic;
            cycle_number      : in unsigned(3 downto 0);
            RMW     : in std_logic;  --read-modify-write instruction
            ACR     : in std_logic;  --carry in from ALU
            BRC     : in std_logic;  --branch flag
            reset   : in std_logic;
            tcstate      : out     std_logic_vector(5 downto 0);
            SYNC, SD1, SD2    : out std_logic;
            VEC1    : out std_logic
      );

end TG;

architecture rtl of TG is
```

```
	-- Build an enumerated type for the state machine
	--type state_type is (s0, s1, s2, s3);
	type state_type is (T0, T1F_T1, T2_T0, T2_3, T2_4, T3_4, T2_5, T3_5,
T4_5,
                                    T2_6, T3_6, T4_6, T5_6, T2_7, T3_7,
T4_7, T5_7, T6_7,
                                    T2_B, T3_B, T1F,
                                    T2_RMW5, T3_RMW5, T4_RMW5, T2_RMW6,
T3_RMW6, T4_RMW6, T5_RMW6,
                                    T2_RMW7, T3_RMW7, T4_RMW7_a,
T5_RMW7_a, T6_RMW7_a,
                                    T4_RMW7_b, T5_RMW7_b);

	-- Register to hold the current state
	signal state   : state_type;

begin

	-- Logic to advance to the next state
	process (clk, reset)
	begin
		if reset = '1' then
			state <= T1F_T1;  --yuchen0513
		elsif (rising_edge(clk)) then
			case state is
				when T0=>
					state <= T1F_T1;
				when T1F_T1=>
				if RMW='0' then --not read-modify-write instruction
					if    cycle_number = 2 then
						state <= T2_T0;
					elsif cycle_number = 3 then
						state <= T2_3;
					elsif cycle_number = 4 then
						state <= T2_4;
					elsif cycle_number = 5 then
						state <= T2_5;
					elsif cycle_number = 6 then
						state <= T2_6;
					elsif cycle_number = 7 then
						state <= T2_7;
					elsif cycle_number = 0 then --input =0 stands
for the branch instruction
						state <= T2_B;
					elsif cycle_number = 1 then
						state <= T1F_T1; --yuchen0513
					end if;

				elsif RMW='1' then  --read-modify-write instruction
					if    cycle_number = 2 then
						state <= T2_T0;
					elsif cycle_number = 5 then
```

```
                state <= T2_RMW5;
            elsif cycle_number = 6 then
                state <= T2_RMW6;
            elsif cycle_number = 7 then
                state <= T2_RMW7;
            end if;
        end if;

        when T2_T0=>
            state <= T1F_T1;
        when T2_3 =>
            state <= T0;
        when T2_4 =>
            state <= T3_4;
        when T3_4 =>
            state <= T0;
        when T2_5 =>
            state <= T3_5;
        when T3_5 =>
            if ACR='1' then    --judge page crossing or not
                state <= T4_5;
            else
                state <= T0;
            end if;
        when T4_5 =>
            state <= T0;
        when T2_6 =>
            state <= T3_6;
        when T3_6 =>
            state <= T4_6;
        when T4_6 =>
            if ACR='1' then  --judge page crossing or not
                state <= T5_6;
            else
                state <= T0;
            end if;
        when T5_6 =>
            state <= T0;
        when T2_7 =>
            state <= T3_7;
        when T3_7 =>
            state <= T4_7;
        when T4_7 =>
            state <= T5_7;
        when T5_7 =>
            state <= T6_7;
        when T6_7 =>
            state <= T0;
        when T2_B =>
            if BRC = '1' then
                state <= T3_B;
            else
```

```
                        state <= T1F;
                    end if;
                when T3_B =>
                    if ACR='1' then  --judge page crossing or not
                        state <= T0;
                    else
                        state <= T1F;
                    end if;
                when T1F =>
                    if    cycle_number = 2 then
                        state <= T2_T0;
                    elsif cycle_number = 3 then
                        state <= T2_3;
                    elsif cycle_number = 4 then
                        state <= T2_4;
                    elsif cycle_number = 5 then
                        state <= T2_5;
                    elsif cycle_number = 6 then
                        state <= T2_6;
                    elsif cycle_number = 7 then
                        state <= T2_7;
                    elsif cycle_number = 0 then --input =0 stands
for the branch instruction
                        state <= T2_B;
                    end if;
                --Read-modify-write instruction
                when T2_RMW5 =>
                    state <= T3_RMW5;
                when T3_RMW5 =>
                    state <= T4_RMW5;
                when T4_RMW5 =>
                    state <= T0;
                when T2_RMW6 =>
                    state <= T3_RMW6;
                when T3_RMW6 =>
                    state <= T4_RMW6;
                when T4_RMW6 =>
                    state <= T5_RMW6;
                when T5_RMW6 =>
                    state <= T0;
                when T2_RMW7 =>
                    state <= T3_RMW7;
                when T3_RMW7 =>
                    if ACR ='1' then state <= T4_RMW7_a;  --page
crossing
                    else state <= T4_RMW7_b; --no page crossing
                    end if;
                when T4_RMW7_a =>
                    state <= T5_RMW7_a;
                when T5_RMW7_a =>
                    state <= T6_RMW7_a;
                when T6_RMW7_a =>
```

```
                        state <= T0;
                    when T4_RMW7_b =>
                        state <= T5_RMW7_b;
                    when T5_RMW7_b =>
                        state <= T0;
                    when others =>
                        state <= T0;
            end case;
      end if;
end process;

-- Output depends solely on the current state
process (state)
begin
      case state is
            when T0 =>
                  tcstate <= "111110";
                  SYNC <= '0';
                  VEC1 <= '0';
                  SD1 <= '0';
                  SD2 <= '0';
            when T1F_T1 =>
                  tcstate <= "111101";
                  SYNC <= '1';
                  VEC1 <= '0';
                  SD1 <= '0';
                  SD2 <= '0';
            when T2_T0 =>
                  tcstate <= "111010";
                  SYNC <= '0';
                  VEC1 <= '0';
                  SD1 <= '0';
                  SD2 <= '0';
            when T2_3 =>
                  tcstate <= "111011";
                  SYNC <= '0';
                  VEC1 <= '0';
                  SD1 <= '0';
                  SD2 <= '0';
            when T2_4 =>
                  tcstate <= "111011";
                  SYNC <= '0';
                  VEC1 <= '0';
                  SD1 <= '0';
                  SD2 <= '0';
            when T3_4 =>
                  tcstate <= "110111";
                  SYNC <= '0';
                  VEC1 <= '0';
                  SD1 <= '0';
                  SD2 <= '0';
            when T2_5 =>
```

```
        tcstate <= "111011";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T3_5 =>
        tcstate <= "110111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T4_5 =>
        tcstate <= "101111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T2_6 =>
        tcstate <= "111011";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T3_6 =>
        tcstate <= "110111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T4_6 =>
        tcstate <= "101111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T5_6 =>
        tcstate <= "011111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T2_7 =>
        tcstate <= "111011";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T3_7 =>
        tcstate <= "110111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
```

```
      SD2 <= '0';
when T4_7 =>
      tcstate <= "101111";
      SYNC <= '0';
      VEC1 <= '0';
      SD1 <= '0';
      SD2 <= '0';
when T5_7 =>
      tcstate <= "011111";
      SYNC <= '0';
      VEC1 <= '0';
      SD1 <= '0';
      SD2 <= '0';
when T6_7 =>
      tcstate <= "111111";
      SYNC <= '0';
      VEC1 <= '1';
      SD1 <= '0';
      SD2 <= '0';
when T2_B =>
      tcstate <= "111011";
      SYNC <= '0';
      VEC1 <= '0';
      SD1 <= '0';
      SD2 <= '0';
when T3_B =>
      tcstate <= "110111";
      SYNC <= '0';
      VEC1 <= '0';
      SD1 <= '0';
      SD2 <= '0';
when T1F =>
      tcstate <= "111111";
      SYNC <= '1';
      VEC1 <= '0';
      SD1 <= '0';
      SD2 <= '0';
when T2_RMW5 =>
      tcstate <= "111011";
      SYNC <= '0';
      VEC1 <= '0';
      SD1 <= '0';
      SD2 <= '0';
when T3_RMW5 =>
      tcstate <= "110111";
      SYNC <= '0';
      VEC1 <= '0';
      SD1 <= '1';
      SD2 <= '0';
when T4_RMW5 =>
      tcstate <= "101111";
      SYNC <= '0';
```

```
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '1';
when T2_RMW6 =>
        tcstate <= "111011";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T3_RMW6 =>
        tcstate <= "110111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T4_RMW6 =>
        tcstate <= "101111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '1';
        SD2 <= '0';
when T5_RMW6 =>
        tcstate <= "011111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '1';
when T2_RMW7 =>
        tcstate <= "111011";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T3_RMW7 =>
        tcstate <= "110111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T4_RMW7_a =>
        tcstate <= "101111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '0';
        SD2 <= '0';
when T5_RMW7_a =>
        tcstate <= "011111";
        SYNC <= '0';
        VEC1 <= '0';
        SD1 <= '1';
        SD2 <= '0';
when T6_RMW7_a =>
```

```
                        tcstate <= "111111";
                        SYNC <= '0';
                        VEC1 <= '0';
                        SD1 <= '0';
                        SD2 <= '1';
                    when T4_RMW7_b =>
                        tcstate <= "101111";
                        SYNC <= '0';
                        VEC1 <= '0';
                        SD1 <= '1';
                        SD2 <= '0';
                    when T5_RMW7_b =>
                        tcstate <= "011111";
                        SYNC <= '0';
                        VEC1 <= '0';
                        SD1 <= '0';
                        SD2 <= '1';
                    when others =>
                        tcstate <= "111110";
                        SYNC <= '0';
                        VEC1 <= '0';
                        SD1 <= '0';
                        SD2 <= '0';

            end case;
        end process;

end rtl;
```

DE2TOP.VHD

```
--
-- DE2 top-level module
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DE2_TOP is

  port (
    -- Clocks

    CLOCK_27,                                           -- 27 MHz
    CLOCK_50,                                           -- 50 MHz
    EXT_CLOCK : in std_logic;                           -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);              -- Push buttons
    SW : in std_logic_vector(17 downto 0);              -- DPDT switches

    -- LED displays

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
       : out std_logic_vector(6 downto 0);              -- (active low)
    LEDG : out std_logic_vector(8 downto 0);            -- Green LEDs (active
high)
    LEDR : out std_logic_vector(17 downto 0);           -- Red LEDs (active high)

    -- RS-232 interface

    UART_TXD : out std_logic;                           -- UART transmitter
    UART_RXD : in std_logic;                            -- UART receiver
```

```
    -- IRDA interface

    IRDA_TXD : out std_logic;                          -- IRDA Transmitter
    IRDA_RXD : in std_logic;                           -- IRDA Receiver

    -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                         -- Low-byte Data Mask
    DRAM_UDQM,                                         -- High-byte Data Mask
    DRAM_WE_N,                                         -- Write Enable
    DRAM_CAS_N,                                        -- Column Address Strobe
    DRAM_RAS_N,                                        -- Row Address Strobe
    DRAM_CS_N,                                         -- Chip Select
    DRAM_BA_0,                                         -- Bank Address 0
    DRAM_BA_1,                                         -- Bank Address 0
    DRAM_CLK,                                          -- Clock
    DRAM_CKE : out std_logic;                          -- Clock Enable

    -- FLASH

    FL_DQ : inout std_logic_vector(7 downto 0);      -- Data bus
    FL_ADDR : out std_logic_vector(21 downto 0);  -- Address bus
    FL_WE_N,                                           -- Write Enable
    FL_RST_N,                                          -- Reset
    FL_OE_N,                                           -- Output Enable
    FL_CE_N : out std_logic;                           -- Chip Enable

    -- SRAM

    SRAM_DQ : inout unsigned(15 downto 0);         -- Data bus 16 Bits
    SRAM_ADDR : out unsigned(17 downto 0);         -- Address bus 18 Bits
    SRAM_UB_N,                                         -- High-byte Data Mask
    SRAM_LB_N,                                         -- Low-byte Data Mask
    SRAM_WE_N,                                         -- Write Enable
    SRAM_CE_N,                                         -- Chip Enable
    SRAM_OE_N : out std_logic;                         -- Output Enable

    -- USB controller

    OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
    OTG_ADDR : out std_logic_vector(1 downto 0);    -- Address
    OTG_CS_N,                                          -- Chip Select
    OTG_RD_N,                                          -- Write
    OTG_WR_N,                                          -- Read
    OTG_RST_N,                                         -- Reset
    OTG_FSPEED,                     -- USB Full Speed, 0 = Enable, Z =
Disable
    OTG_LSPEED : out std_logic;     -- USB Low Speed, 0 = Enable, Z =
Disable
    OTG_INT0,                                          -- Interrupt 0
```

```
    OTG_INT1,                                       -- Interrupt 1
    OTG_DREQ0,                                      -- DMA Request 0
    OTG_DREQ1 : in std_logic;                       -- DMA Request 1
    OTG_DACK0_N,                                    -- DMA Acknowledge 0
    OTG_DACK1_N : out std_logic;                    -- DMA Acknowledge 1

    -- 16 X 2 LCD Module

    LCD_ON,                     -- Power ON/OFF
    LCD_BLON,                   -- Back Light ON/OFF
    LCD_RW,                     -- Read/Write Select, 0 = Write, 1 = Read
    LCD_EN,                     -- Enable
    LCD_RS : out std_logic;     -- Command/Data Select, 0 = Command, 1 =
Data
    LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

    -- SD card interface

    SD_DAT : in std_logic;      -- SD Card Data      SD pin 7 "DAT
0/DataOut"
    SD_DAT3 : out std_logic;    -- SD Card Data 3    SD pin 1 "DAT 3/nCS"
    SD_CMD : out std_logic;     -- SD Card Command   SD pin 2 "CMD/DataIn"
    SD_CLK : out std_logic;     -- SD Card Clock     SD pin 5 "CLK"

    -- USB JTAG link

    TDI,                        -- CPLD -> FPGA (data in)
    TCK,                        -- CPLD -> FPGA (clk)
    TCS : in std_logic;         -- CPLD -> FPGA (CS)
    TDO : out std_logic;        -- FPGA -> CPLD (data out)

    -- I2C bus

    I2C_SDAT : inout std_logic; -- I2C Data
    I2C_SCLK : out std_logic;   -- I2C Clock

    -- PS/2 port

    PS2_DAT,                    -- Data
    PS2_CLK : in std_logic;     -- Clock

    -- VGA output

    VGA_CLK,                                -- Clock
    VGA_HS,                                 -- H_SYNC
    VGA_VS,                                 -- V_SYNC
    VGA_BLANK,                              -- BLANK
    VGA_SYNC : out std_logic;               -- SYNC
    VGA_R,                                  -- Red[9:0]
    VGA_G,                                  -- Green[9:0]
    VGA_B : out unsigned(9 downto 0);       -- Blue[9:0]
```

```
    --  Ethernet Interface

    ENET_DATA : inout unsigned(15 downto 0);    -- DATA bus 16 Bits
    ENET_CMD,           -- Command/Data Select, 0 = Command, 1 = Data
    ENET_CS_N,                                  -- Chip Select
    ENET_WR_N,                                  -- Write
    ENET_RD_N,                                  -- Read
    ENET_RST_N,                                 -- Reset
    ENET_CLK : out std_logic;                   -- Clock 25 MHz
    ENET_INT : in std_logic;                    -- Interrupt

    -- Audio CODEC

    AUD_ADCLRCK : inout std_logic;                      -- ADC LR Clock
    AUD_ADCDAT : in std_logic;                          -- ADC Data
    AUD_DACLRCK : inout std_logic;                      -- DAC LR Clock
    AUD_DACDAT : out std_logic;                         -- DAC Data
    AUD_BCLK : inout std_logic;                         -- Bit-Stream Clock
    AUD_XCK : out std_logic;                            -- Chip Clock

    -- Video Decoder

    TD_DATA : in std_logic_vector(7 downto 0);  -- Data bus 8 bits
    TD_HS,                                      -- H_SYNC
    TD_VS : in std_logic;                       -- V_SYNC
    TD_RESET : out std_logic;                   -- Reset

    -- General-purpose I/O

    GPIO_0,                                      -- GPIO Connection 0
    GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
    );

end DE2_TOP;

architecture datapath of DE2_TOP is
signal Databus, DOR, ROM_data, X, Y : std_logic_vector(7 downto 0);--
yuchen0514
signal Addrbus, ROM_address  : std_logic_vector(15 downto 0);
signal W_R : std_logic;
signal reset : std_logic; --JB0513
signal Sclk : std_logic; --JB0513
signal clk25 : std_logic := '0'; --yuchen0514

component SixFiveO2
port(
           Databus :in std_logic_vector(7 downto 0);
           Addrbus :out std_logic_vector(15 downto 0);
           DOR , P, X_Reg_out, Y_Reg_out : out std_logic_vector(7 downto 0);
      reset, clk   :in std_logic;
           XL, XH, YL, YH, ACCL, ACCH  : out std_logic_vector(6 downto 0);
           W_R     : out std_logic);
```

```
end component;

component rom is
port(addr  : in std_logic_vector(15 downto 0);
           data : out std_logic_vector(7 downto 0));
end component;

component SRAMCtrl is
port (
           reset, clk, W_R : in  std_logic;
           ROM_data, DOR  : in std_logic_vector(7 downto 0);
           databus    : out std_logic_vector(7 downto 0);
           AddressBus : in std_logic_vector(15 downto 0);
           ROM_address : out std_logic_vector(15 downto 0);
           SRAM_DQ    : inout unsigned(15 downto 0);
           SRAM_ADDR  : out unsigned(17 downto 0);
           SRAM_UB_N,            -- High-byte Data Mask
           SRAM_LB_N,            -- Low-byte Data Mask
           SRAM_WE_N,            -- Write Enable
           SRAM_CE_N,            -- Chip Enable
           SRAM_OE_N : out std_logic  -- Output Enable
);
end component;

component debounce --JB0513
     port (
           clk, resetsw : in std_logic;
           resetout  : out std_logic
           );
end component debounce;

component slowclk --JB0513
     port (
           clkin : in std_logic;
           clkout  : out std_logic
           );
end component slowclk;

component de2_vga_raster is

  port (
    reset : in std_logic;
    clk   : in std_logic;                    -- Should be 25.125 MHz

    center: in std_logic_vector(15 downto 0) := X"f0f0"; -- circle center

    VGA_CLK,                          -- Clock
    VGA_HS,                           -- H_SYNC
    VGA_VS,                           -- V_SYNC
    VGA_BLANK,                        -- BLANK
    VGA_SYNC : out std_logic;         -- SYNC
    VGA_R,                            -- Red[9:0]
```

```
    VGA_G,                                -- Green[9:0]
    VGA_B : out unsigned(9 downto 0) -- Blue[9:0]
    );

end component;

begin

  --HEX2     <= (others => '1');
  HEX1     <= (others => '1');
  HEX0     <= (others => '1');          -- Rightmost
  LEDG(8)     <= '0';
  LEDR     <= (others => '0');
  LCD_ON   <= '1';
  LCD_BLON <= '1';
  LCD_RW <= '1';
  LCD_EN <= '0';
  LCD_RS <= '0';


  SD_DAT3 <= '1';
  SD_CMD <= '1';
  SD_CLK <= '1';


  UART_TXD <= '0';
  DRAM_ADDR <= (others => '0');
  DRAM_LDQM <= '0';
  DRAM_UDQM <= '0';
  DRAM_WE_N <= '1';
  DRAM_CAS_N <= '1';
  DRAM_RAS_N <= '1';
  DRAM_CS_N <= '1';
  DRAM_BA_0 <= '0';
  DRAM_BA_1 <= '0';
  DRAM_CLK <= '0';
  DRAM_CKE <= '0';
  FL_ADDR <= (others => '0');
  FL_WE_N <= '1';
  FL_RST_N <= '0';
  FL_OE_N <= '1';
  FL_CE_N <= '1';
  OTG_ADDR <= (others => '0');
  OTG_CS_N <= '1';
  OTG_RD_N <= '1';
  OTG_RD_N <= '1';
  OTG_WR_N <= '1';
  OTG_RST_N <= '1';
  OTG_FSPEED <= '1';
  OTG_LSPEED <= '1';
  OTG_DACK0_N <= '1';
  OTG_DACK1_N <= '1';
```

```
  TDO <= '0';

  I2C_SCLK <= '0';

  IRDA_TXD <= '0';

  ENET_CMD <= '0';
  ENET_CS_N <= '1';
  ENET_WR_N <= '1';
  ENET_RD_N <= '1';
  ENET_RST_N <= '1';
  ENET_CLK <= '0';

  AUD_DACDAT <= '0';
  AUD_XCK <= '0';

  TD_RESET <= '0';

  -- Set all bidirectional ports to tri-state
  DRAM_DQ     <= (others => 'Z');
  FL_DQ       <= (others => 'Z');
  OTG_DATA    <= (others => 'Z');
  LCD_DATA    <= (others => 'Z');
  I2C_SDAT    <= 'Z';
  ENET_DATA   <= (others => 'Z');
  AUD_ADCLRCK <= 'Z';
  AUD_DACLRCK <= 'Z';
  AUD_BCLK    <= 'Z';
  GPIO_0      <= (others => 'Z');
  GPIO_1      <= (others => 'Z');


--JB0513: port map below changed to fit in debounce

debouncecode: debounce port map(clk=>CLOCK_50, resetsw=>SW(17),
resetout=>reset); --JB0513
slowclkcode: slowclk port map(clkin=>CLOCK_50, clkout=>Sclk); --JB0513

CPUConnect: SixFiveO2 port map(clk=>Sclk, reset=>reset, W_R=>W_R, XH=>HEX7,
XL=>HEX6, YH=>HEX5, YL=>HEX4, ACCH=>HEX3, ACCL=>HEX2,
                                                    X_Reg_out=>X,
Y_Reg_out=>Y,    Databus=>Databus, DOR=>DOR, Addrbus=>Addrbus,P=>LEDG(7
downto 0));

InstructionROM: Rom port map(addr=>ROM_address, data=>ROM_data);

MemorySRAM: SRAMCtrl port map(reset=>reset, clk=>Sclk, W_R=>W_R,
ROM_data=>ROM_data, DOR=>DOR,
                                                  databus=>databus,
AddressBus=>Addrbus, ROM_address=>ROM_address,
                                                  SRAM_DQ=>SRAM_DQ,
```

```
                                                        SRAM_ADDR=>SRAM_ADDR,
                                                        SRAM_UB_N=>SRAM_UB_N,
                                                        SRAM_LB_N=>SRAM_LB_N,
                                                        SRAM_WE_N=>SRAM_WE_N,
                                                        SRAM_CE_N=>SRAM_CE_N,
                                                       SRAM_OE_N=>SRAM_OE_N);

VGAClock:
process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
      clk25 <= not clk25;
    end if;
end process;

VGA:  de2_vga_raster port map (reset => reset, clk => clk25,
                                                          VGA_CLK => VGA_CLK,
                                                          VGA_HS => VGA_HS,
                                                       VGA_VS => VGA_VS,
                                                          VGA_BLANK =>
VGA_BLANK,
                                                        VGA_SYNC => VGA_SYNC,
                                                          VGA_R => VGA_R,
                                                          VGA_G => VGA_G,
                                                          VGA_B => VGA_B,
                                                          center(15 downto 8)
=> X,
                                                          center(7 downto
0)=>Y
  );
end datapath;
```

CPU.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity  CPU is
port(
            clk, SD1, SD2, reset, VEC1     : in std_logic;
            opcode : in std_logic_vector(7 downto 0);
            tcstate : in std_logic_vector(5 downto 0);
            databus : in std_logic_vector(7 downto 0);
            ACR_out, W_R, BRC      : out std_logic; --JB0510 added BRC as CPU
output
            ABL_out, ABH_out, DOR, X_out, Y_out, ACC_out, P_out : out
std_logic_vector(7 downto 0)
            );
end CPU;

architecture rtl of CPU is
      signal ABL, ABH : std_logic_vector(7 downto 0);
      signal X, Y, ACC, S, AI, BI, ADD, P : unsigned(7 downto 0);
      signal PC      : unsigned(15 downto 0) := (others=>'0');
      signal SUMS, I_ADDC, ORS, ANDS, EORS, SRS   : std_logic;
      --signal opcode : std_logic_vector(7 downto 0);
      signal ACR    : std_logic;
      --signal temp : unsigned(8 downto 0);
      signal Mask_shortcut :std_logic;
      --signal counter : unsigned(7 downto 0);
      signal proceed : std_logic; --JB0513
begin
      ACR_out<=ACR;


--ALU Part: Combinational Logic
      process(SUMS, ORS, ANDS, EORS, SRS, AI, BI, I_ADDC, Mask_shortcut,
opcode, P, ACR)

      variable temp : unsigned(8 downto 0)   ;
--    variable ACR  : std_logic;
```

```
      begin


      if SUMS='1' then
                              temp :=('0'&AI)+('0'&BI)+("0" & I_ADDC);
                              ADD<=temp(7 downto 0);
                              ACR<=temp(8) or Mask_shortcut;
      elsif ORS='1'  then ADD<=AI or BI;  ACR<='0'; temp:="000000000";
      elsif ANDS='1' then ADD<=AI and BI; ACR<='0'; temp:="000000000";
      elsif EORS='1' then ADD<=AI xor BI; ACR<='0'; temp:="000000000";
      elsif SRS='1'  then ADD(6 downto 0)<=BI(7 downto 1); ACR<=BI(0);
temp:="000000000";
            if opcode(7 downto 5)="010" then ADD(7)<='0';
            elsif opcode(7 downto 5)="011" then ADD(7)<=P(0);
            else  ADD<=x"00"; ACR<='0';
            end if;
      else ADD<=x"00"; ACR<='0'; temp:="000000000";
      end if;

      end process;

--=============================================================================
--=============================================================================
      process(clk, reset)
      begin

      if rising_edge(clk) then --JB0513 reset if statement is put into
rising_edge(clk)

      if reset = '1' then PC<=x"0001"; ABL<=x"00"; ABH<=x"00"; X<=x"00";
Y<=x"00";
                                       ACC<=x"11"; AI<=x"00"; BI<=x"00";
S<=x"00"; DOR<=x"00";
                                       SUMS<='0'; ORS<='0'; ANDS<='0';
EORS<='0'; SRS<='0';
                                       I_ADDC<='0'; W_R<='1'; SUMS<='1';
Mask_shortcut<='0'; P<=x"00";
                                       --counter<=x"00";
      elsif reset = '0' then

            if opcode=x"00" then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
            ABH<=std_logic_vector(PC(15 Downto 8));
            end if;

            --JB0513 STOP CODE: hex FF
            if opcode=x"FF" then
                  PC <= PC;
                  ABL <= ABL;
                  ABH <= ABH;
            end if;
```

```
            ----------------------------Yu's code starts here-----------------
-----------
            --======================bbb is the only
concern=========================
            if not(opcode(1 downto 0)="00") then --exclude the cc=00 part to
avoid overlapping
            --Address Mode: Absolute; aaa: don't care; cc: don't care.

                  --Timing: T2
                  if (opcode(4 downto 2)="011" and tcstate(2)='0' ) then
                        PC<=PC+1;
                        ABL<=std_logic_vector(PC(7 Downto 0));
                        ABH<=std_logic_vector(PC(15 Downto 8));
                        Sums<='1';
                        AI<=x"00";
                        BI<=unsigned(Databus);
                        I_ADDC<='0';
                  end if;

                  --Timing: T3
                  if (opcode(4 downto 2)="011" and tcstate(3)='0' ) then
                        PC<=PC;
                        ABL<=std_logic_vector(ADD);
                        ABH<=databus;
                        SUMS<='0';
                  end if;

                  --Address Mode: Zero page, X; aaa: don't care; cc: don't
care.
                  --Timing T2
                  if (opcode(4 downto 2)="101" and tcstate(2)='0' and
(not((opcode(7 downto 5)="100" or opcode(7 downto 5)="101") and opcode(1
downto 0)="10"))) then
                        PC<=PC;
                        ABL<=Databus;
                        ABH<=x"00";
                        AI<=unsigned(X);
                        BI<=unsigned(Databus);
                        Sums<='1';
                  end if;

                  --Timing T3
                  if (opcode(4 downto 2)="101" and tcstate(3)='0' ) then
                        PC<=PC;
                        ABL<=std_logic_vector(ADD);
                        ABH<=x"00";
                        SUMS<='0';
                  end if;

                 --Address Mode: Absolute X; aaa: don't care; cc: don't care.
                  --Timing T2
```

```
                if (opcode(4 downto 2)="111" and tcstate(2)='0' and
(not(opcode(7 downto 5)="101" and opcode(1 downto 0)="10"))) then
                      PC<=PC+1;
                      ABL<=std_logic_vector(PC(7 Downto 0));
                      ABH<=std_logic_vector(PC(15 Downto 8));
                      AI<=unsigned(X);
                      BI<=unsigned(Databus);
                      Sums<='1';

                      if opcode(7 downto 5)="100" then
                            Mask_shortcut<='1';
                      end if;
                end if;

                --Timing T3
                --no page crossing
                if (opcode(4 downto 2)="111" and tcstate(3)='0' and
ACR='0' ) then
                      PC<=PC;
                      ABL<=std_logic_vector(ADD);
                      ABH<=Databus;
                      Mask_shortcut<='0';
                      Sums<='0';
                end if;
                --page crossing
                if (opcode(4 downto 2)="111" and tcstate(3)='0' and
ACR='1' ) then
                      PC<=PC;
                      ABL<=std_logic_vector(ADD);
                      ABH<=x"00";
                      AI<=x"00";
                      BI<=unsigned(Databus);
                      I_ADDC<='1';
                      Sums<='1';
                      Mask_shortcut<='0';
                end if;

                --Timing T4
                if (opcode(4 downto 2)="111" and tcstate(4)='0' ) then
                      PC<=PC;
                      ABL<=ABL;
                      ABH<=std_logic_vector(ADD);
                      I_ADDC<='0';
                      SUMS<='0';
                end if;

                --Address Mode: Zero Page; aaa: don't care; cc: don't care.
                --Timing T2
                if (opcode(4 downto 2)="001" and tcstate(2)='0' ) then
                      PC<=PC;
                      ABL<=Databus;
                      ABH<=x"00";
```

```
                end if;

            end if; --exclude the cc=00 part to avoid overlapping
            --======================bbb is the only concern
ends====================

            --======================bbb and cc=01 are
concerned==============================
            --Address Mode: Absolute Y; aaa: don't care; cc: 01
            --Timing T2
            if (opcode(4 downto 2)="110" and opcode(1 downto 0)="01" and
tcstate(2)='0' ) then
                PC<=PC+1;
                ABL<=std_logic_vector(PC(7 Downto 0));
                ABH<=std_logic_vector(PC(15 Downto 8));
                AI<=unsigned(Y);
                BI<=unsigned(Databus);
                Sums<='1';
                if opcode(7 downto 5)="100" then Mask_shortcut<='1';
                end if;
            end if;

            --Timing T3
            --no page crossing
            if (opcode(4 downto 2)="110" and opcode(1 downto 0)="01" and
tcstate(3)='0' and ACR='0' ) then
                PC<=PC;
                ABL<=std_logic_vector(ADD);
                ABH<=Databus;
                Sums<='0';
                Mask_shortcut<='1';
            end if;
            --page crossing
            if (opcode(4 downto 2)="110" and opcode(1 downto 0)="01" and
tcstate(3)='0' and ACR='1' ) then
                PC<=PC;
                ABL<=std_logic_vector(ADD);
                ABH<=x"00";
                AI<=x"00";
                BI<=unsigned(Databus);
                I_ADDC<='1';
                Sums<='1';
                Mask_shortcut<='1';
            end if;

            --Timing T4
            if (opcode(4 downto 2)="110" and opcode(1 downto 0)="01" and
tcstate(4)='0' ) then
                PC<=PC;
                ABL<=ABL;
                ABH<=std_logic_vector(ADD);
                I_ADDC<='0';
```

```
                Sums<='0';
            end if;

            --Address Mode: (Zero page, X)/Indirect, X; aaa: don't care; cc:
01
            --T2
            if (opcode(4 downto 2)="000" and opcode(1 downto 0)="01" and
tcstate(2)='0' ) then
                PC<=PC;
                ABH<=x"00";
                ABL<=Databus;
                AI<=X;
                BI<=unsigned(Databus);
                Sums<='1';
            end if;

            --T3
            if (opcode(4 downto 2)="000" and opcode(1 downto 0)="01" and
tcstate(3)='0' ) then
                PC<=PC;
                ABH<=x"00";
                ABL<=std_logic_vector(ADD);
                AI<=x"00";
                BI<=ADD;
                I_ADDC<='1';
                Sums<='1';
                Mask_shortcut<='1';
            end if;

            --T4
            if (opcode(4 downto 2)="000" and opcode(1 downto 0)="01" and
tcstate(4)='0' ) then
                PC<=PC;
                ABH<=x"00";
                ABL<=std_logic_vector(ADD);
                AI<=x"00";
                BI<=unsigned(Databus);
                Sums<='1';
                I_ADDC<='0';
                Mask_shortcut<='0';
            end if;

            --T5
            if (opcode(4 downto 2)="000" and opcode(1 downto 0)="01" and
tcstate(5)='0' ) then
                PC<=PC;
                ABH<=Databus;
                ABL<=std_logic_vector(ADD);
                Sums<='0';
            end if;
```

```
            --Address Mode: (Zero page), Y/Indirect, Y; aaa: don't care; cc:
01
            --T2
            if (opcode(4 downto 2)="100" and opcode(1 downto 0)="01" and
tcstate(2)='0' ) then
                  PC<=PC;
                  ABH<=x"00";
                  ABL<=Databus;
                  AI<=x"00";
                  BI<=unsigned(Databus);
                  Sums<='1';
            I_ADDC<='1';
            end if;

            --T3
            if (opcode(4 downto 2)="100" and opcode(1 downto 0)="01" and
tcstate(3)='0' ) then
                  PC<=PC;
                  ABH<=x"00";
                  ABL<=std_logic_vector(ADD);
                  AI<=Y;
                  BI<=unsigned(Databus);
                  Sums<='1';
                  I_ADDC<='0';
                  if opcode(7 downto 5)="100" then
                        Mask_shortcut<='1';
                  end if;
            end if;

            --T4
            --no page crossing
            if (opcode(4 downto 2)="100" and opcode(1 downto 0)="01" and
tcstate(4)='0' and ACR='0') then
                  PC<=PC;
                  ABH<=Databus;
                  ABL<=std_logic_vector(ADD);
                  Mask_shortcut<='0';
                  Sums<='0';
            end if;

            --page crossing
            if (opcode(4 downto 2)="100" and opcode(1 downto 0)="01" and
tcstate(4)='0' and ACR='1') then
                  PC<=PC;
                  ABH<=Databus;
                  ABL<=std_logic_vector(ADD);
                  AI<=x"00";
                  BI<=unsigned(Databus);
                  Sums<='1';
                  I_ADDC<='1';
                  Mask_shortcut<='0';
            end if;
```

```
            --T5
            if (opcode(4 downto 2)="100" and opcode(1 downto 0)="01" and
tcstate(5)='0' ) then
                  PC<=PC;
                  ABH<=std_logic_vector(ADD);
                  ABL<=ABL;
                  I_ADDC<='0';
                  Sums<='0';
            end if;
            --=======================bbb and cc=01 are concerned
ends============================

            --=======================aaa and cc=01 are
concerned==================================
            --Instruction: LDA; aaa: 101; bbb: don't care; cc: 01
            --T0
            if (opcode(7 downto 5)="101" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  ACC<=unsigned(Databus);
            end if;

            --T1
            if (opcode(7 downto 5)="101" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  P(7)<=ACC(7);
                  SUMS<='0';
                  if ACC=x"00" then
                        P(1)<='1';
                  else
                        P(1)<='0';
                  end if;
            end if;

      --Instruction: ORA; aaa: 000; bbb: don't care; cc: 01
            --T0
            if (opcode(7 downto 5)="000" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  AI<=ACC;
                  BI<=unsigned(Databus);
                  ORS<='1';
                  SUMS<='0';
            end if;
```

```
            --T1
            if (opcode(7 downto 5)="000" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  ACC<=ADD;
                  ORS<='0';
                  if ADD(7)='1' then
                        P(7)<='1';
                  else
                        P(7)<='0';
                  end if;

                  if ADD=x"00" then
                        P(1)<='1';
                  else
                        P(1)<='0';
                  end if;
            end if;

      --Instruction: AND; aaa: 001; bbb: don't care; cc: 01
            --T0
            if (opcode(7 downto 5)="001" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  AI<=ACC;
                  BI<=unsigned(Databus);
                  ANDS<='1';
                  SUMS<='0';
            end if;

            --T1
            if (opcode(7 downto 5)="001" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  ACC<=ADD;
                  ANDS<='0';
                  if ADD(7)='1' then
                        P(7)<='1';
                  else
                        P(7)<='0';
                  end if;

                  if ADD=x"00" then
                        P(1)<='1';
                  else
```

```
                    P(1)<='0';
                end if;
            end if;

            --Instruction: EOR; aaa: 010; bbb: don't care; cc: 01
            --T0
            if (opcode(7 downto 5)="010" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
                PC<=PC+1;
                ABL<=std_logic_vector(PC(7 Downto 0));
                ABH<=std_logic_vector(PC(15 Downto 8));
                AI<=ACC;
                BI<=unsigned(Databus);
                EORS<='1';
                SUMS<='0';
            end if;

            if (opcode(7 downto 5)="010" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
                PC<=PC+1;
                ABL<=std_logic_vector(PC(7 Downto 0));
                ABH<=std_logic_vector(PC(15 Downto 8));
                ACC<=ADD;
                EORS<='0';

                if ADD(7)='1' then
                    P(7)<='1';
                else
                    P(7)<='0';
                end if;

                if ADD=x"00" then
                    P(1)<='1';
                else
                    P(1)<='0';
                end if;
            end if;

      --Instruction: ADC; aaa: 011; bbb: don't care; cc: 01
            --T0
            if (opcode(7 downto 5)="011" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
                PC<=PC+1;
                ABL<=std_logic_vector(PC(7 Downto 0));
                ABH<=std_logic_vector(PC(15 Downto 8));
                AI<=ACC;
                BI<=unsigned(Databus);
                SUMS<='1';
                I_ADDC<=P(0);
            end if;

            --T1
```

```
            if (opcode(7 downto 5)="011" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  ACC<=ADD;
                  SUMS<='0';
                  I_ADDC<='0';

                  if ADD(7)='1' then
                        P(7)<='1';
                  else
                        P(7)<='0';
                  end if;

                  if ADD=x"00" then
                        P(1)<='1';
                  else
                        P(1)<='0';
                  end if;

                  if ACR='1' then
                        P(0)<='1';
                  else
                        P(0)<='0';
                  end if;
            end if;


      --Instruction: CMP; aaa: 110; bbb: don't care; cc: 01
            --T0
            if (opcode(7 downto 5)="110" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  AI<=ACC;
                  BI<=not(unsigned(Databus)); --yuchen0513
                  I_ADDC<='1';
                  SUMS<='1';
            end if;

            --T1
            if (opcode(7 downto 5)="110" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  --ACC<=ADD;  --yuchen0513
                  I_ADDC<='0';
                  SUMS<='0';
                  if ADD(7)='1' then
```

```
                        P(7)<='1';
                    else
                        P(7)<='0';
                    end if;

                    if ADD=x"00" then
                        P(1)<='1';
                    else
                        P(1)<='0';
                    end if;
                    P(0)<=ACR;
                end if;

        --Instruction: SBC; aaa: 111; bbb: don't care; cc: 01
                --T0
                if (opcode(7 downto 5)="111" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
                    PC<=PC+1;
                    ABL<=std_logic_vector(PC(7 Downto 0));
                    ABH<=std_logic_vector(PC(15 Downto 8));
                    AI<=ACC;
                    BI<=unsigned(not(Databus));
                    I_ADDC<=P(0);
                    SUMS<='1';
                end if;

                --T1
                if (opcode(7 downto 5)="111" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
                    PC<=PC+1;
                    ABL<=std_logic_vector(PC(7 Downto 0));
                    ABH<=std_logic_vector(PC(15 Downto 8));
                    ACC<=ADD;
                    SUMS<='0';
                    I_ADDC<='0';

                    if ADD(7)='1' then
                        P(7)<='1';
                    else
                        P(7)<='0';
                    end if;

                    if ADD=x"00" then
                        P(1)<='1';
                    else P(1)<='0';
                    end if;

                    if ACR='0' then
                        P(0)<='1';
                    else
                        P(0)<='0';
                    end if;
```

```
            end if;


            --Instruction: STA; aaa: 100; bbb: don't care; cc: 01
            --T0
            if (opcode(7 downto 5)="100" and opcode(1 downto 0)="01" and
tcstate(0)='0' ) then
            PC<=PC+1;
            ABL<=std_logic_vector(PC(7 Downto 0));
            ABH<=std_logic_vector(PC(15 Downto 8));
            W_R<='1';
            end if;

            --T1
            if (opcode(7 downto 5)="100" and opcode(1 downto 0)="01" and
tcstate(1)='0' ) then
            PC<=PC+1;
            ABL<=std_logic_vector(PC(7 Downto 0));
            ABH<=std_logic_vector(PC(15 Downto 8));
            end if;

            --Last cycle, write value to memory
            if (opcode(7 downto 5)="100" and opcode(1 downto 0)="01") then
                  if  ((opcode(4 downto 2)="000" and tcstate(5)='0') --(zero
page, X)
                        or(opcode(4 downto 2)="001" and tcstate(2)='0') --zero
page
                        or(opcode(4 downto 2)="011" and tcstate(3)='0') --
absolute
                        or(opcode(4 downto 2)="100" and ((tcstate(4)='0') or
(tcstate(5)='0'))) --(zero page), Y
                        or(opcode(4 downto 2)="101" and tcstate(3)='0') --zero
page, X
                        or(opcode(4 downto 2)="110" and ((tcstate(3)='0') or
(tcstate(4)='0'))) --absolute, Y
                        or(opcode(4 downto 2)="111" and ((tcstate(3)='0') or
(tcstate(4)='0')))) --absolute, X
                  then
                        DOR<=std_logic_vector(ACC); W_R<='0';
                  end if;
            end if;
            --=======================aaa and cc=01 are concerned
ends==============================

            --=======================cc=10 are concerned
=========================================
            --STX and LDX address mode special cases
            --1) Address Mode: Zero page, X==> Zero page, Y; Instructions:
STX and LDX
            --Timing T2
```

```
            if (opcode(4 downto 2)="101" and tcstate(2)='0' and (((opcode(7
downto 5)="100" or opcode(7 downto 5)="101") and opcode(1 downto 0)="10")))
then
                  PC<=PC;
                  ABL<=Databus;
                  ABH<=x"00";
                  AI<=unsigned(Y);
                  BI<=unsigned(Databus);
                  Sums<='1';
            end if;
            --2) Address Mode: absolute X==> absolute Y; Instruction: LDX
            --Timing T2
            if (opcode(4 downto 2)="111" and tcstate(2)='0' and ((opcode(7
downto 5)="101" and opcode(1 downto 0)="10"))) then
                  PC<=PC+1;
                  ABL<=std_logic_vector(PC(7 Downto 0));
                  ABH<=std_logic_vector(PC(15 Downto 8));
                  AI<=unsigned(Y);
                  BI<=unsigned(Databus);
                  Sums<='1';
            end if;



          --JB0511 To avoid conflict with Arthy's, avoid hex1>=8 and hex2=A.
            if opcode(1 downto 0)="10" and not(opcode(7)='1' and opcode(3
downto 2)="10") then

                  --For all the instructions in this section (cc=10), T0 and
T1 has the same behaviors
                  --T0
                  if tcstate(0)='0' then
                        PC<=PC+1;
                        ABL<=std_logic_vector(PC(7 Downto 0));
                        ABH<=std_logic_vector(PC(15 Downto 8));
                        W_R<='1';
                        --LDX needs one more operation
                        if opcode(7 downto 5)="101" then
                              X<=unsigned(Databus);
                        end if;
                  end if;

                  --T1
                  if tcstate(1)='0' then
                        PC<=PC+1;
                        ABL<=std_logic_vector(PC(7 Downto 0));
                        ABH<=std_logic_vector(PC(15 Downto 8));
                        --LDX affects the flags
                        if opcode(7 downto 5)="101" then
                              if X(7)='1' then P(7)<='1';
                              else P(7)<='0';
```

```
                        end if;
                        if X=x"00" then P(1)<='1';
                        else P(1)<='0';
                        end if;
                  end if;
            end if;

      --Instruction: ASL; aaa: 000; bbb: don't care; cc: 10
            --SD1
            if (opcode(7 downto 5)="000" and SD1='1') then
                  PC<=PC;
                  ABH<=ABH;
                  ABL<=ABL;
                  AI<=unsigned(Databus); --JB0511 correct? Why is W_R
'0' while Databus=>DOR?
                  BI<=unsigned(Databus);
                  DOR<=Databus;
                  SUMS<='1';
                  W_R<='0';
                  I_ADDC<='0';
            end if;

      --Instruction: ROL; aaa: 001; bbb: don't care; cc: 10
            --SD1
            if (opcode(7 downto 5)="001" and SD1='1') then
                  PC<=PC;
                  ABH<=ABH;
                  ABL<=ABL;
                  AI<=unsigned(Databus);
                  BI<=unsigned(Databus);
                  Sums<='1';
                  I_ADDC<=P(0);
                  DOR<=Databus;
                  W_R<='0';
            end if;

            --Instruction: LSR; aaa: 010; bbb: don't care; cc: 10
            --SD1
            if (opcode(7 downto 5)="010" and SD1='1') then
                  PC<=PC;
                  ABH<=ABH;
                  ABL<=ABL;
                  BI<=unsigned(Databus);
                  AI<=x"00";
                  DOR<=std_logic_vector(ADD);
                  SUMS<='0';
                  SRS<='1';
                  W_R<='0';
                  --ADD(7)<='0';
            end if;

            --SD2
```

```
                if (opcode(7 downto 5)="010" and SD2='1') then
                      PC<=PC;
                      ABH<=ABH;
                      ABL<=ABL;
                      DOR<=std_logic_vector(ADD);
                      I_ADDC<='0';
                      W_R<='0';
                      SRS<='0';
                      if ADD=x"00" then
                            P(1)<='1';
                      else
                            P(1)<='0';
                      end if;

                      if ACR='1' then
                            P(0)<='1';
                      else
                      P(0)<='0';
                      end if;
                end if;

          --Instruction: ROR; aaa: 011; bbb: don't care; cc: 10
                --SD1
                if (opcode(7 downto 5)="011" and SD1='1') then
                      PC<=PC;
                      ABH<=ABH;
                      ABL<=ABL;
                      BI<=unsigned(Databus);
                      AI<=x"00";
                      DOR<=Databus;
                      SRS<='1';
                      W_R<='0';
                      SUMS<='0';
                      --ADD(7)<=P(0);
                end if;

                --Instruction ASL or ROL or ROR
                --SD2
                if ((opcode(7 downto 5)="011" or opcode(7 downto 5)="000"
or opcode(7 downto 5)="001") and SD2='1') then
                      PC<=PC;
                      ABH<=ABH;
                      ABL<=ABL;
                      DOR<=std_logic_vector(ADD);
                      W_R<='0';
                      SRS<='0';
                      SUMS<='0';

                      if ADD(7)='1' then
                            P(7)<='1';
                      else
                            P(7)<='0';
```

```
            end if;

            if ADD=x"00" then
                  P(1)<='1';
            else
                  P(1)<='0';
            end if;

            if ACR='1' then
                  P(0)<='1';
            else
                  P(0)<='0';
            end if;
      end if;

--Instruction: INC; aaa: 111; bbb: don't care; cc: 10
      --SD1
      if (opcode(7 downto 5)="111" and SD1='1') then
            PC<=PC;
            ABH<=ABH;
            ABL<=ABL;
            BI<=unsigned(Databus);
            AI<=x"00";
            I_ADDC<='1';
            SUMS<='1';
            DOR<=Databus;
            W_R<='0';
      end if;

      --SD2
      if (opcode(7 downto 5)="111" and SD2='1') then
            PC<=PC;
            ABH<=ABH;
            ABL<=ABL;
            DOR<=std_logic_vector(ADD);
            W_R<='0';
            I_ADDC<='0';
            SUMS<='0';
            if ADD(7)='1' then
                  P(7)<='1';
            else
                  P(7)<='0';
            end if;
            if ADD=x"00" then
                  P(1)<='1';
            else
                  P(1)<='0';
            end if;
      end if;


      --Instruction: DEC; aaa: 110; bbb: don't care; cc: 10
```

```
                --SD1
                if (opcode(7 downto 5)="110" and SD1='1') then
                      PC<=PC;
                      ABH<=ABH;
                      ABL<=ABL;
                      BI<=unsigned(Databus);
                      AI<=x"ff";
                      I_ADDC<='0';
                      DOR<=Databus;
                      W_R<='0'; --JB0510 W_R=0 signal should happen at the
following cycle, not when DOR is loaded??
                      SUMS<='1';
                end if;

                --SD2
                if (opcode(7 downto 5)="110" and SD2='1') then
                      PC<=PC;
                      ABH<=ABH;
                      ABL<=ABL;
                      DOR<=std_logic_vector(ADD);
                      W_R<='0'; --JB0510 W_R=0 signal should happen at the
following cycle, not when DOR is loaded??
                      I_ADDC<='0';
                      SUMS<='0';

                      if ADD(7)='1' then P(7)<='1';
                            else P(7)<='0';
                      end if;

                      if ADD=x"00" then P(1)<='1';
                            else P(1)<='0';
                      end if;
                end if;

                --Instruction: STX; aaa: 100; bbb: don't care; cc: 10

                --Last cycle, write value to memory
                if opcode(7 downto 5)="100" then
                      if  ((opcode(4 downto 2)="001" and tcstate(2)='0') --
zero page
                            or(opcode(4 downto 2)="011" and tcstate(3)='0')
--absolute
                            or(opcode(4 downto 2)="101" and tcstate(3)='0'))
--zero page, Y
                      then
                            DOR<=std_logic_vector(X);
                            W_R<='0'; --JB0510 W_R=0 signal should happen at
the following cycle, not when DOR is loaded??
                      end if;
                end if;
```

```
                --Address mode: accumulator cc:10 --JB0511 this lies inside
not(hex1>=8 and hex2=A) to distinguish from Arthy's.
                if opcode(4 downto 2)="010"  then
                      --Instruction: ASL aaa=000
                      --T2+T0
                      if (opcode(7 downto 5)="000" and tcstate="111010")
then
                            PC<=PC;
                            ABL<=ABL;
                            ABH<=ABH;
                            AI<=ACC;
                            BI<=ACC;
                            I_ADDC<='0';
                            SUMS<='1';
                      end if;

                      --Instruction: ROL aaa=001
                      --T2+T0
                      if (opcode(7 downto 5)="001" and tcstate="111010")
then
                            PC<=PC;
                            ABL<=ABL;
                            ABH<=ABH;
                            AI<=ACC;
                            BI<=ACC;
                            I_ADDC<=P(0);
                            SUMS<='1';
                      end if;

                --Instruction: LSR or ROR aaa=010 or 011
                      --T2+T0
                      if ((opcode(7 downto 5)="010" or opcode(7 downto
5)="011") and tcstate="111010") then
                            PC<=PC;
                            ABL<=ABL;
                            ABH<=ABH;
                            BI<=ACC;
                            AI<=x"00";
                            SUMS<='0';
                            SRS<='1';
                      end if;

                --Instruction: ASL or ROL or LSR or ROR
                      --T1
                      if tcstate(1)='0' then
                            PC<=PC+1;
                            ABL<=std_logic_vector(PC(7 Downto 0));
                            ABH<=std_logic_vector(PC(15 Downto 8));
                            ACC<=ADD;
                            SUMS<='0';
                            I_ADDC<='0'; --JB0512
                            W_R<='1'; --JB0512
```

```
                              SRS<='0';
                              if ADD(7)='1' then P(7)<='1';
                              else P(7)<='0';
                              end if;
                              if ADD=x"00" then P(1)<='1';
                              else P(1)<='0';
                              end if;
                              if ACR='1' then P(0)<='1';
                              else P(0)<='0';
                              end if;
                        end if;
                  end if;
            end if;
            ---==cc=10 AND aaa=0xx are concerned
ends=========================================
            -------------YU's code starts here-----------------------------
-----

            -------------JAEBIN's code starts here--------------------------
---------

            --cc=00
            if (opcode(1 downto 0)="00") then

                  ----------branch: xxy10000---------------------------------
--------------
                  if (opcode(4 downto 2)="100") then --bbb=100 does not
overlab with anything else.
                        --T2
                        if (tcstate(2)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              BI<=unsigned(Databus);
                              AI<=PC(7 Downto 0);
                              Sums<='1';
                        end if;

                        --T1
                        if tcstate="111111" then   --yuchen0514
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              sums<='0';
                        end if;

                        --T3
                        if (tcstate(3)='0') then
                              ABL<=std_logic_vector(ADD);
                              PC(7 downto 0)<=ADD+1;
                              Sums<='0';
                        end if;
```

```
                        --T0
                        if (tcstate(0)='0') then
                              PC(15 downto 8)<=PC(15 downto 8)+1;
                              ABH<=std_logic_vector(PC(15 downto 8)+1);
                        end if;

                  ----------------branch done--------------------------------
---------

                  -----------interrupts: 0xx00000----------------------------
------------
                  --four total, JSR, RTS, BRK, RTI.
                  elsif (opcode(4 downto 2)="000" and opcode(7)='0') then

                        --1/4. JSR abs. hex:20
                        if (opcode(6 downto 5)="01") then
                              --T2
                              if (tcstate(2)='0') then
                                    S<=unsigned(Databus);
                                    ABL<=std_logic_vector(S);
                                    ABH<=x"01";
                                    BI<=S;
                                    AI<=x"00";
                                    Sums<='1';
                              end if;
                              --T3
                              if (tcstate(3)='0') then
                                    ABL<=std_logic_vector(ADD);
                                    ABH<=x"01";
                                    DOR<=std_logic_vector(PC(15 Downto 8));
                                    BI<=ADD;
                                    AI<=x"ff";
                                    Sums<='1';
                              end if;
                              --T4
                              if (tcstate(4)='0') then
                                    --Databus<=DOR;
                                    W_R<='0';
                                    ABL<=std_logic_vector(ADD);
                                    ABH<=x"01";
                                    DOR<=std_logic_vector(PC(7 Downto 0));
                                    BI<=ADD;
                                    AI<=x"ff";
                                    Sums<='1';
                              end if;
                              --T5
                              if (tcstate(5)='0') then
                                    --Databus<=DOR;
                                    W_R<='0';
                                    PC<=PC+1;
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
```

```
                BI<=ADD;
                AI<=x"00";
                Sums<='1';
            end if;
            --T0
            if (tcstate(0)='0') then
                W_R<='1';
                ABL<=std_logic_vector(S);
                ABH<=Databus;
                PC<=(unsigned(Databus) & S) + 1;
                S<=ADD;
                Sums<='0';
            end if;
            --T1
            if (tcstate(1)='0') then
                PC<=PC+1;
                ABL<=std_logic_vector(PC(7 Downto 0));
                ABH<=std_logic_vector(PC(15 Downto 8));
                Sums<='0';
            end if;
        end if;

        ----2/4. RTS. hex:60
        if (opcode(6 downto 5)="11") then
            --T2
            if (tcstate(2)='0') then
                ABL<=std_logic_vector(S);
                ABH<=x"01";
                BI<=S;
                AI<=x"00";
                I_ADDC<='1';
                Sums<='1';
            end if;
            --T3
            if (tcstate(3)='0') then
                ABL<=std_logic_vector(ADD);
                ABH<=x"01";
                BI<=ADD;
                AI<=x"00";
                I_ADDC<='1';
                Sums<='1';
            end if;
            --T4
            if (tcstate(4)='0') then
                ABL<=std_logic_vector(ADD);
                ABH<=x"01";
                S<=ADD;
                I_ADDC<='0';
                Sums<='0';
            end if;
            --T5
            if (tcstate(5)='0') then
```

```
                ABL<=std_logic_vector(PC(7 downto 0));
                ABH<=Databus;
                PC<=(unsigned(Databus) & PC(7 downto 0))+1;
                Sums<='0';
            end if;
            --T0
            if (tcstate(0)='0') then
                PC<=PC+1;
                ABL<=std_logic_vector(PC(7 Downto 0));
                ABH<=std_logic_vector(PC(15 Downto 8));
                Sums<='0';
            end if;
            --T1
            if (tcstate(1)='0') then
                PC<=PC+1;
                ABL<=std_logic_vector(PC(7 Downto 0));
                ABH<=std_logic_vector(PC(15 Downto 8));
                Sums<='0';
            end if;
        end if;

        ----4/4. RTI. hex:40
        if (opcode(6 downto 5)="10") then
            --T2
            if (tcstate(2)='0') then
                ABL<=std_logic_vector(S);
                ABH<=x"01";
                BI<=S;
                AI<=x"00";
                I_ADDC<='1';
                Sums<='1';
            end if;
            --T3
            if (tcstate(3)='0') then
                ABL<=std_logic_vector(ADD);
                ABH<=x"01";
                BI<=ADD;
                AI<=x"00";
                I_ADDC<='1';
                Sums<='1';
            end if;
            --T4
            if (tcstate(4)='0') then
                P<=unsigned(Databus);
                ABL<=std_logic_vector(ADD);
                ABH<=x"01";
                BI<=ADD;
                AI<=x"00";
                I_ADDC<='1';
                Sums<='1';
            end if;
            --T5
```

```
                        if (tcstate(5)='0') then
                              PC(7 downto 0)<=unsigned(Databus);
                              ABL<=std_logic_vector(ADD);
                              ABH<=x"01";
                              S<=ADD;
                              I_ADDC<='0';
                              Sums<='0';
                        end if;
                        --T0
                        if (tcstate(0)='0') then
                              ABL<=std_logic_vector(PC(7 downto 0));
                              ABH<=Databus;
                              PC<=(unsigned(Databus) & PC(7 downto 0))+1;
                              Sums<='0';
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              Sums<='0';
                        end if;
                  end if;




            ---------the rest of cc=00, excluding branch and interrupt-
---------------
            --no overlap with Arthy's, because Arthy's only have
bbb=010 and bbb=110.
            else

                  --1.bbb=000 AND aaa=1xx. immediate.
                  if (opcode(4 downto 2)="000" and opcode(7)='1') then
                        --T0
                        if (tcstate(2)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              Sums<='0';
                        end if;
                  end if;

                  --2.bbb=001. zeropage. common to all aaa within cc=00
```

```
                        if (opcode(4 downto 2)="001") then
                              --T2
                              if (tcstate(2)='0') then
                                    ABL<=Databus;
                                    ABH<=x"00";
                                    Sums<='0';
                              end if;
                              --both T0 and T1
                              if (tcstate(0)='0'or tcstate(1)='0') then
                                    PC<=PC+1;
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
                                    Sums<='0';
                              end if;
                        end if;

                        ---3.bbb=011. absolute. exceptions to JMP ABS(aaa=010)
and JMP IND(aaa=011)
                        ---exception 1/2: JMP ABS. bbb=011, aaa=010
                        if (opcode(4 downto 2)="011" and opcode(7 downto
5)="010") then
                              --T2
                              if (tcstate(2)='0') then
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
                                    PC(7 Downto 0)<=unsigned(Databus);
                                    Sums<='0';
                              end if;
                              --T0
                              if (tcstate(0)='0') then
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=Databus;
                                    PC<=(unsigned(Databus) & PC(7 downto 0))+1;
                                    Sums<='0';
                              end if;
                              --T1
                              if (tcstate(1)='0') then
                                    PC<=PC+1;
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
                                    Sums<='0';
                              end if;
                        end if;

                        ---exception 2/2: JMP IND. bbb=011, aaa=011
                        if (opcode(4 downto 2)="011" and opcode(7 downto
5)="011") then
                              --both T2 and T4
                              if (tcstate(2)='0' or tcstate(4)='0') then
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
                                    PC(7 Downto 0)<=unsigned(Databus);
```

```
                              Sums<='0';
                        end if;
                        --both T3 and T0
                        if (tcstate(3)='0' and tcstate(0)='0') then
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=Databus;
                              PC<=(unsigned(Databus) & PC(7 downto 0))+1;
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                        end if;
                  end if;

                  ---the rest of bbb=011 absolute.
                  if (opcode(4 downto 2)="011" and not(opcode(7 downto
5)="010") and not(opcode(7 downto 5)="011")) then
                        --T2
                        if (tcstate(2)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              BI<=unsigned(Databus);
                              AI<=x"00";
                              Sums<='1';
                        end if;
                        --T3
                        if (tcstate(3)='0') then
                              ABH<=Databus;
                              ABL<=std_logic_vector(ADD);
                              Sums<='0';
                        end if;
                        --T0
                        if (tcstate(0)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              Sums<='0';
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              Sums<='0';
                        end if;
                  end if;

                  ---4.bbb=101. zeropage,X. common to all aaa within
cc=00
```

```
                        if (opcode(4 downto 2)="101") then
                              --T2
                              if (tcstate(2)='0') then
                                    BI<=unsigned(Databus);
                                    AI<=unsigned(X);
                                    Sums<='1';
                              end if;
                              --T3
                              if (tcstate(3)='0') then
                                    ABL<=std_logic_vector(ADD);
                                    ABH<=x"00";
                                    Sums<='0';
                              end if;
                              --T0
                              if (tcstate(0)='0') then
                                    PC<=PC+1;
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
                                    Sums<='0';
                              end if;
                              --T1
                              if (tcstate(1)='0') then
                                    PC<=PC+1;
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
                                    Sums<='0';
                              end if;
                        end if;

                        ---5.bbb=111. absolute,X. common to all aaa within
cc=00
                        if (opcode(4 downto 2)="111") then
                              --T2
                              if (tcstate(2)='0') then
                                    PC<=PC+1;
                                    ABL<=std_logic_vector(PC(7 Downto 0));
                                    ABH<=std_logic_vector(PC(15 Downto 8));
                                    BI<=unsigned(Databus);
                                    AI<=unsigned(X);
                                    Sums<='1';
                                    if opcode(7 downto 5)="100" then
Mask_shortcut<='1';--Yuchen
                                    end if;
                              end if;
                              --T3
                              if (tcstate(3)='0') then
                                    ABH<=Databus;
                                    ABL<=std_logic_vector(ADD);
                                    BI<=unsigned(Databus);
                                    I_ADDC<=ACR;
                                    Mask_shortcut<='0';--Yuchen
                              end if;
```

```
                        --T4
                        if (tcstate(4)='0') then
                              ABH<=std_logic_vector(ADD);
                              Sums<='0';
                        end if;
                        --T0
                        if (tcstate(0)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                              Sums<='0';
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              PC<=PC+1;
                              ABL<=std_logic_vector(PC(7 Downto 0));
                              ABH<=std_logic_vector(PC(15 Downto 8));
                        end if;
                  end if;

                  ------------bbb taken care of. now aaa.---------------
------------------

                  ---1.aaa=101. LDY. common to all bbb within cc=00
                  if (opcode(7 downto 5)="101") then
                        --T0
                        if (tcstate(0)='0') then
                              Y<=unsigned(Databus);
                              BI<=unsigned(Databus);
                              AI<=x"00";
                              Sums<='1';
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              P(7)<=ADD(7); --JB set N. P is the
processor status register (1 byte)
                              if ADD=x"00" then P(1)<='1';
                              else P(1)<='0';
                              end if;
                        end if;
                  end if;

                  ---2.aaa=111. CPX. common to all bbb within cc=00
                  if (opcode(7 downto 5)="111") then
                        --T0
                        if (tcstate(0)='0') then
                              BI<=not(unsigned(Databus)); --JB0511 need
to invert!!!!!!!!!!!!
                              AI<=unsigned(X);
                              Sums<='1';
                              I_ADDC<='1';  --yuchen0513
                        end if;
```

```
                        --T1
                        if (tcstate(1)='0') then
                              P(7)<=ADD(7);
                              if ADD=x"00" then P(1)<='1';
                              else P(1)<='0';
                              end if;
                              P(0)<=ACR;
                              Sums<='0';
                              I_ADDC<='0';  --yuchen0513
                        end if;
                  end if;

                  ---3.aaa=110. CPY. common to all bbb within cc=00
                  if (opcode(7 downto 5)="110") then
                        --T0
                        if (tcstate(0)='0') then
                              BI<=unsigned(not(Databus)); --JB0511 need
to invert!!!!!!!!!!!!
                              AI<=unsigned(Y);
                              Sums<='1';
                              I_ADDC<='1';  --yuchen0513
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              P(7)<=ADD(7);
                              if ADD=x"00" then P(1)<='1';
                              else P(1)<='0';
                              end if;
                              P(0)<=ACR;
                              Sums<='0';
                              I_ADDC<='0';  --yuchen0513
                        end if;
                  end if;

                  ---4.aaa=001. BIT. common to all bbb within cc=00
                  if (opcode(7 downto 5)="001") then
                        --T0
                        if (tcstate(0)='0') then
                              BI<=unsigned(Databus);
                              AI<=ACC;
                              Sums<='1';
                        end if;
                        --T1
                        if (tcstate(1)='0') then
                              P(7)<=ADD(7);
                              if ADD=x"00" then P(1)<='1';
                              else P(1)<='0';
                              end if;
                              P(0)<=ACR;
                              Sums<='0';
                        end if;
                  end if;
```

```
                        ---5.aaa=100. STY. common to all bbb within cc=00
                        if (opcode(7 downto 5)="100") then
                              --both T2 and T3
                              if (tcstate(2)='0' or tcstate(3)='0') then --JB
Y->DOR happenes at T2 in zeropage, and at T3 in the rest.
                                    DOR<=std_logic_vector(Y);
                              end if;
                              --T0
                              if (tcstate(0)='0') then
                                    --Databus<=DOR;
                                    W_R<='0';
                              end if;
                              --T1
                              if (tcstate(0)='0') then
                                    Sums<='0';
                                    W_R<='1';
                              end if;
                        end if;

                  ----------------------------aaa taken care of.-------------
--------------------
                  end if;

            end if;
            ------------------------------JAEBIN's code ends here-------------
-----------------
```

```
-------------------------ARTHY's code starts here-------------------------
--
            --==========================SINGLE BYTE
INSTRUCTIONS====BEGIN==========
            --NOP
            if(opcode(7 downto 0) = x"EA") then
                --if(tcstate(2) = '0') then --JB0511 deleted "and tcstate(0)
= '0'"
                        --PC <= PC; -- this also not required
                 --end if;
                 if (tcstate(1) = '0') then
                        PC <=PC+1;
```

```
                        ABL<=std_logic_vector(PC(7 Downto 0));
                        ABH<=std_logic_vector(PC(15 Downto 8));
                  end if;
            end if; --NOP ends.



            --PHA/PLA/PHP/PLP
            if((opcode(7 downto 0) = x"48") or (opcode(7 downto 0) = x"68")
or (opcode(7 downto 0) = x"08") or
            (opcode(7 downto 0) = x"28"))  then --PHA/PLA/PHP/PLP

                  --T2
                  if(tcstate(2)='0') then
                        PC <= PC - 1; --JB0511 PC-1?????
                        ABL<=std_logic_vector(S(7 downto 0) );
                        ABH<= x"01";
                        BI <= S;
                        SUMS<='1';

                        -- Push PHA / PHP
                        if(opcode(5) = '0') then --subtract.
                              AI <= x"ff";
                              W_R <= '0';
                              if(opcode(6) = '1') then
                                    -- PHA put the acc onto databus
                                    DOR <= std_logic_vector(ACC);
                              else
                                    -- (opcode(6) == '0') then
                                    -- PHP put the status reg unto db

                                    DOR <= std_logic_vector(P);
                              end if;

                        -- Pull PLA / PLP
                        else --sum.
                              AI <= x"01";
                        end if;
                  end if;

                  --T3
                  if(tcstate(3) = '0') then -- assume only PLA and PLP get
here
                        W_R <= '1'; -- back to read
                        PC<=PC;
                        S<=ADD;
                        SUMS <= '0';
                        ABL<=std_logic_vector(ADD);
                        ABH<=x"01";
                  end if;

                  --T0
```

```
                if (tcstate(0)= '0') then
                      ABL<=std_logic_vector(PC(7 Downto 0));
                      ABH<=std_logic_vector(PC(15 Downto 8));
                      PC <= PC + 1;

                      --PLA/PLP
                      if(opcode(5) = '1') then

                            --PLA
                            if(opcode(6) = '1') then
                                  ACC <= unsigned(Databus);
                                  if (ACC = 0) then
                                        P(1) <= '1'; --set zero flag
                                  end if;
                                  P(7) <= ACC(7);  -- set negative flag
                            --PLP
                            elsif(opcode(6) = '0') then
                                  P <= unsigned(Databus);
                            end if;

                      --PHA/PHP
                      elsif(opcode(5) = '0') then
                            --SUBS <= '1'; --JB0511 Subtracts WHAT? Nothing
goes in to AI or BI.
                            S<= ADD;
                            W_R <= '1'; -- read
                      end if;
                end if;

                --T1
                if (tcstate(1)= '0') then -- PHA/PHP/PLA/PLP
                      ABL<=std_logic_vector(PC(7 Downto 0));
                      ABH<=std_logic_vector(PC(15 Downto 8));
                      PC <= PC + 1;
                end if;

          end if; --PHA/PLA/PHP/PLP end.


          -- INX, INY, DEX, DEY
          --1/4 DEX: CA
          if opcode (7 downto 0) = x"CA" then
                --T2+T0
                if (tcstate(2) = '0') then
                      SUMS <= '1';
                      BI <= X; AI <= x"ff";
                end if;
                --T1
                if (tcstate(1) = '0') then
                      ABL<=std_logic_vector(PC(7 Downto 0));
                      ABH<=std_logic_vector(PC(15 Downto 8));
                      PC <= PC + 1;
```

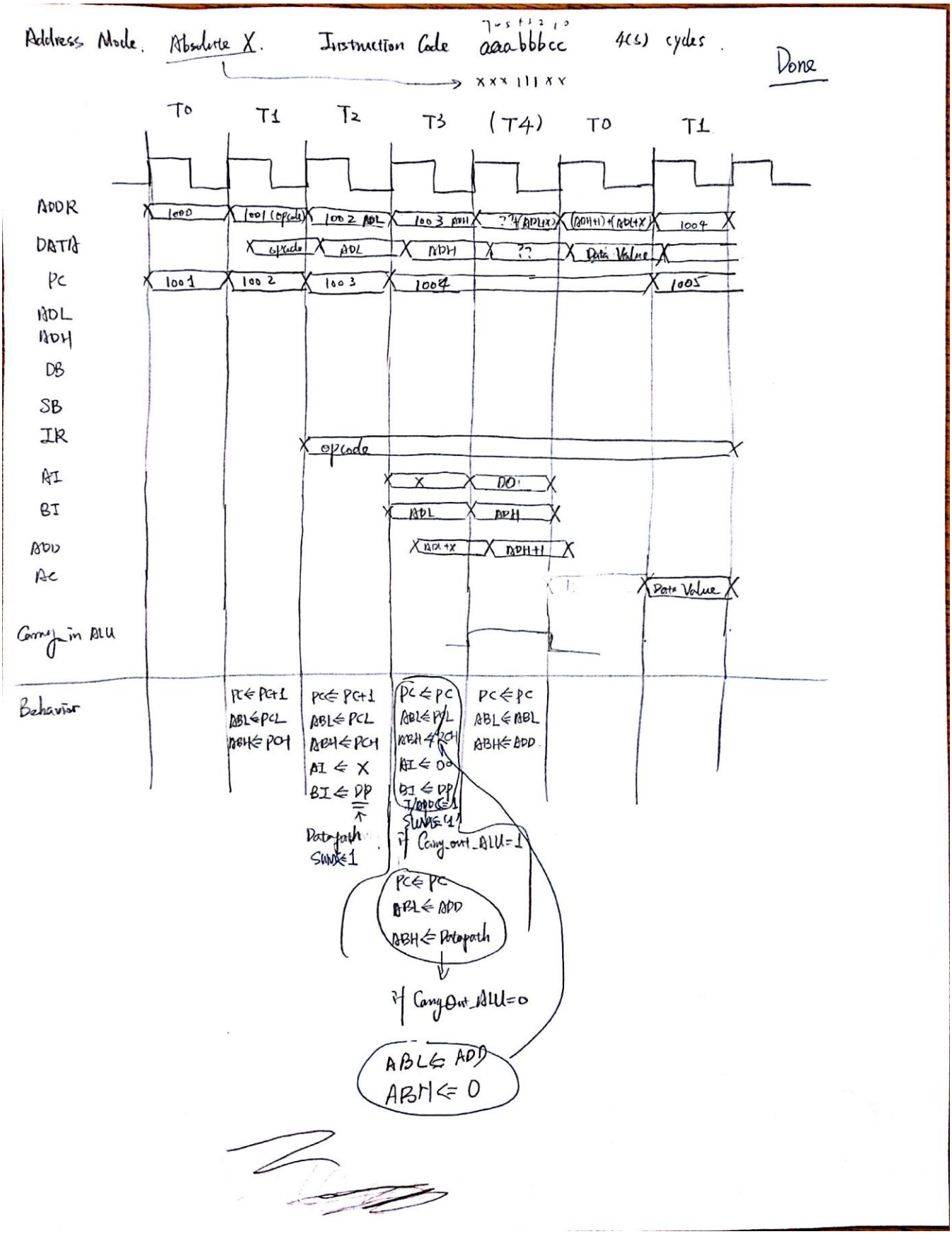
```
            if (ADD = 0) then
                  P(1) <= '1'; --Z flag
            end if;
            P(7) <= ADD(7); --N flag

            X <= ADD;
            SUMS <= '0';
      end if;
end if;
--2/4 INX: E8
if opcode (7 downto 0) = x"E8" then
      --T2+T0
      if (tcstate(2) = '0') then
            SUMS <= '1';
            BI <= X; AI <= x"01";
            I_ADDC<='0';  --yuchen0514
      end if;
      --T1
      if (tcstate(1) = '0') then
            ABL<=std_logic_vector(PC(7 Downto 0));
            ABH<=std_logic_vector(PC(15 Downto 8));
            PC <= PC + 1;
            if (ADD = 0) then
                  P(1) <= '1'; --Z flag
            end if;
            P(7) <= ADD(7); --N flag

            X <= ADD;
            SUMS <= '0';
            I_ADDC<='0';  --yuchen0514
      end if;
end if;
--3/4 DEY: 88
if opcode (7 downto 0) = x"88" then
      --T2+T0
      if (tcstate(2) = '0') then
            SUMS <= '1';
            BI <= Y; AI <= x"ff";
      end if;
      --T1
      if (tcstate(1) = '0') then
            ABL<=std_logic_vector(PC(7 Downto 0));
            ABH<=std_logic_vector(PC(15 Downto 8));
            PC <= PC + 1;
            if (ADD = 0) then
                  P(1) <= '1'; --Z flag
            end if;
            P(7) <= ADD(7); --N flag

            Y <= ADD;
            SUMS <= '0';
      end if;
```

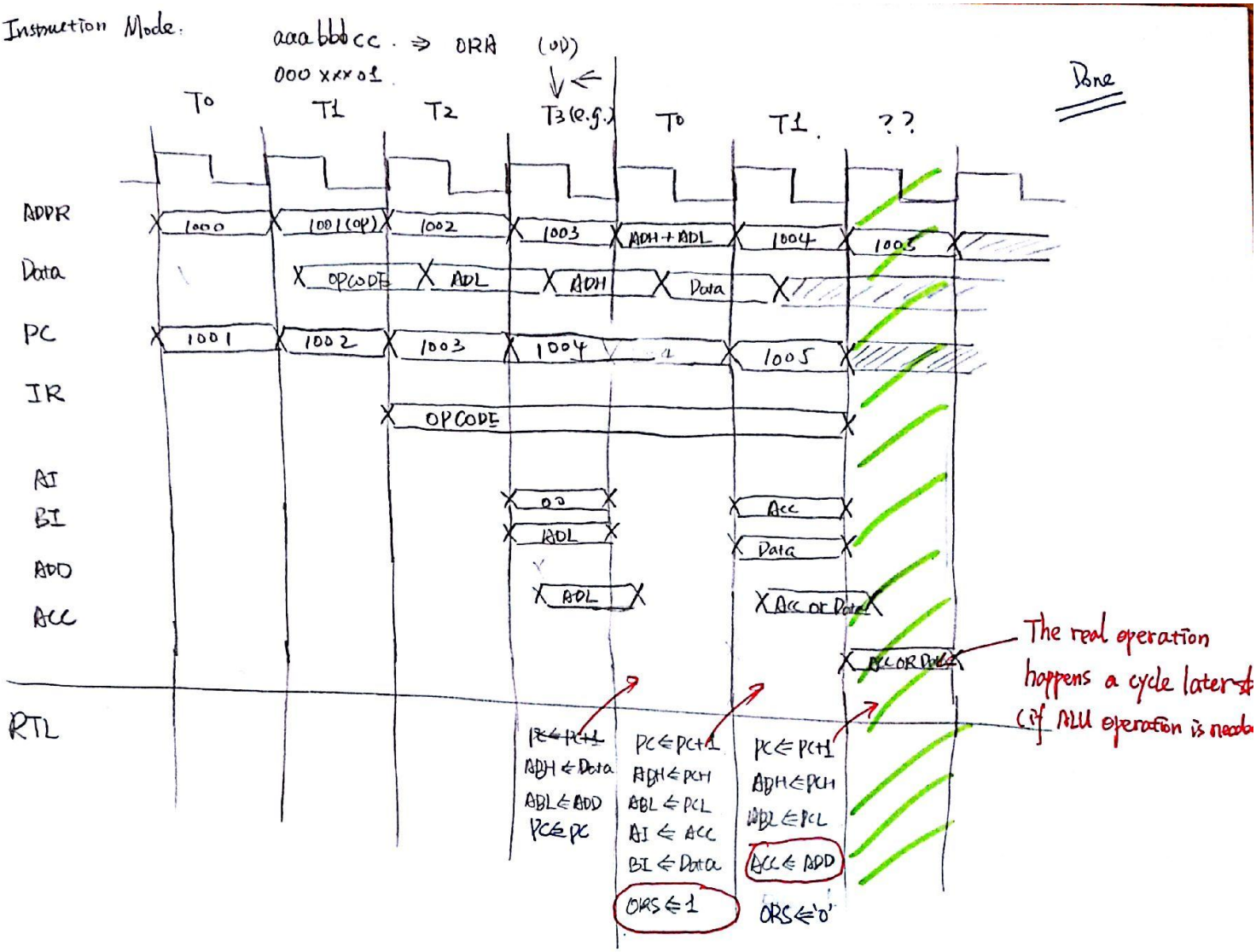
```
            end if;
            --4/4 INY: C8
            if opcode (7 downto 0) = x"c8" then --yuchen0514
                  --T2+T0
                  if (tcstate(2) = '0') then
                        SUMS <= '1';
                        BI <= Y; AI <= x"01";
                        I_ADDC<='0';  --yuchen0514
                  end if;
                  --T1
                  if (tcstate(1) = '0') then
                        ABL<=std_logic_vector(PC(7 Downto 0));
                        ABH<=std_logic_vector(PC(15 Downto 8));
                        PC <= PC + 1;
                        if (ADD = 0) then
                              P(1) <= '1'; --Z flag
                        end if;
                        P(7) <= ADD(7); --N flag

                        Y <= ADD;
                        SUMS <= '0';
                        I_ADDC<='0';  --yuchen0514
                  end if;
            end if;
            -- INX, INY, DEX, DEY ends.


            --Register instructions
            if opcode(4 downto 2) = "110" and opcode(0) = '0' then

                  --T2 + T0
                  if tcstate(2) = '0' then --JB0511 deleted "and tcstate(0) =
'0'"
                        --PC <= PC;
                        CASE opcode(7 downto 5)IS
                              when "000" => P(0) <= '0'; -- CLC
                              when "001" => P(0) <= '1'; -- SEC
                              when "010" => P(2) <= '0'; ---CLI
                              when "011" => P(2) <= '1'; -- SEI
                              when "101" => P(6) <= '0'; -- CLV
                              when "110" => P(3) <= '0'; -- CLD
                              when "111" => P(3) <= '1'; -- SED
                              when others => null;
                        END CASE;
                  end if;

                  --T1
                  if (tcstate(1) = '0') then
                        ABL<=std_logic_vector(PC(7 Downto 0));
                        ABH<=std_logic_vector(PC(15 Downto 8));
                        PC <= PC + 1;
                  end if;
```

```
            end if; --Register instructions end.


            -- Transfer instructions
           if ((opcode (7 downto 0) = x"8A") or (opcode (7 downto 0) = x"9A")
or
                 (opcode (7 downto 0) = x"AA") or (opcode (7 downto 0) =
x"BA") or
                 (opcode (7 downto 0) = x"98") or (opcode (7 downto 0) =
x"A8")) then

                 --T2 + T0
                 if(tcstate(2) = '0') then --JB0511 removed "and
tcstate(0)='0'"
                       --PC <= PC;
                       if((opcode (7 downto 0)) = x"8A") then -- TXA
                            ACC <= X;
                            if (X = 0) then
                                  P(1) <= '1';
                            end if;
                            P(7) <= X(7);
                       end if;
                       if ((opcode (7 downto 0)) = x"9A") then --TXS
                            S <= X;
                       end if;
                       if ((opcode (7 downto 0)) = x"AA" ) then -- TAX
                            X <= ACC;
                            if (ACC = 0) then
                                  P(1) <= '1';
                            end if;
                            P(7) <= ACC(7);
                       end if;
                       if ((opcode (7 downto 0)) = x"BA") then -- TSX
                            X <= S;
                            if (S = 0) then
                                  P(1) <= '1';
                            end if;
                            P(7) <= S(7);
                       end if;
                       if ((opcode (7 downto 0)) = x"98") then --TYA
                            ACC <= Y;
                            if (Y = 0) then
                                  P(1) <= '1';
                            end if;
                            P(7) <= Y(7);
                       end if;
                       if ((opcode (7 downto 0)) = x"A8") then -- TAY
                            Y <= ACC; --TAY
                            if (ACC = 0) then
                                  P(1) <= '1';
                            end if;
```

```
                        P(7) <= ACC(7);
                    end if;
                end if;
                --T1
                if (tcstate(1) = '0') then
                        ABL<=std_logic_vector(PC(7 Downto 0));
                        ABH<=std_logic_vector(PC(15 Downto 8));
                        PC <= PC + 1;
                end if;
            end if; -- Transfer instructions end.

            ----------------------------Arthy's code ends here--------------
----------------------


     end if; --clk rising edge
end if; --reset
end process;

--branch conditions are judged combinationally so that it can supply
information to TG on time.
process(opcode, P)
begin
     if (opcode(1 downto 0)="00") then

                ---------------------------------branch: xxy10000-----------
-------------------------------------
                if (opcode(4 downto 2)="100") then --bbb=100 does not
overlab with anything else.
                        -- xx=00. N flag. P(7)
                        if (  (opcode(7 downto 6)="00" and
P(7)=opcode(5)) or -- xx=00. N flag. P(7)
                                        (opcode(7 downto 6)="01" and
P(6)=opcode(5)) or -- xx=01. V(O) flag. P(6)
                                        (opcode(7 downto 6)="10" and
P(0)=opcode(5)) or -- xx=10. C flag. P(0)
                                        (opcode(7 downto 6)="11" and
P(1)=opcode(5)))    -- xx=11. Z flag. P(1)
                        then
                            BRC<='1';
                        else BRC<='0';  --yuchen 0514
                        end if;
               else BRC<='0';  --yuchen 0514
                end if;
     else BRC<='0';  --yuchen 0514
     end if;
end process;


process(ABH, ABL, ACC, X, Y, P)
begin
           ABH_out<=ABH;
```

```
            ABL_out<=ABL;
            ACC_out<=std_logic_vector(ACC);
            X_out<=std_logic_vector(X);
            Y_out<=std_logic_vector(Y);
            P_out<=std_logic_vector(P);
      end process;
end rtl;
```


Address Mode. Absolute X. Instruction Code aaabbbcc 4(5) cycles.
Done
T0 T1 T2 T3 (T4) T0 T1
ADDR
DATA
PC
ADL
ADH
DB
SB
IR
AI
BI
ADD
AC
Carry in ALU
Behavior
opcode
Data Value
PC ← PC+1
ABL ← PCL
ABH ← PCH
AI ← X
BI ← DB
Datapath SWDB ← 1
PC ← PC
ABL ← ADD
ABH ← Datapath
Carry_out_ALU=1
Carry_out_ALU=0
ABL ← ADD
ABH ← 0


Instruction Mode:
aaabbbcc ⇒ ORA (OD)
000 xxx 01
Done
T0
T1
T2
T3 (e.g.)
T0
T1
??
ADDR
Data
PC
IR
AI
BI
ADD
ACC
RTL
1000
1001 (OP)
1002
1003
ADH+ADL
1004
1005
OPCODE
ADL
ADH
Data
1001
1002
1003
1004
1005
OPCODE
00
ADL
ADL
Acc
Data
Acc or Data
Acc or Data
The real operation happens a cycle later (if ALU operation is needed)
PC⇐PC+1
ADH⇐Data
ADL⇐ADD
PC⇐PC
PC⇐PC+1
ADH⇐PCH
ADL⇐PCL
AI⇐ACC
BI⇐Data
ORS⇐1
PC⇐PC+1
ADH⇐PCH
ADL⇐PCL
ACC⇐ADD
ORS⇐'0'