Assembly Languages II

Prof. Stephen A. Edwards

with contributions from Prof. Brian Evans, Niranjan Damera-Venkata and Magesh Valliappan, UT Austin

Last Time

- General model of assembly language
  - Undifferentiated sequence of instructions
  - Arithmetic instructions (ADD, SUB)
  - Control-flow (JMP, CALL, RET)
- Four main types: CISC, RISC, DSP, and VLIW
  - CISC
    - Few, special-purpose registers
    - Complex addressing modes
    - Powerful instruction (e.g., string move)
  - RISC
    - Many general-purpose registers
    - Few addressing modes
    - Arithmetic operations don’t touch memory
    - Simple instructions

Digital Signal Processor Apps.

- Low-cost embedded systems
  - Modems, cellular telephones, disk drives, printers
- High-throughput applications
  - Halftoning, base stations, 3-D sonar, tomography
- PC based multimedia
  - Compression/decompression of audio, graphics, video
- Embedded processor requirements
  - Inexpensive with small area and volume
  - Deterministic interrupt service routine latency
  - Low power: ~50 mW (TMS320C54x uses 0.36 µA/MIPS)

Conventional DSP Architecture

- Harvard architecture
  - Separate data memory/bus and program memory/bus
  - Three reads and one or two writes per instruction cycle
  - Deterministic interrupt service routine latency
  - Multiply-accumulate in single instruction cycle
  - Special addressing modes supported in hardware
  - Bit-reversed addressing for fast Fourier transforms
  - Instructions to keep the pipeline (3-4 stages) full
  - Zero-overhead looping (one pipeline flush to set up)
  - Delayed branches

Conventional DSPs

<table>
<thead>
<tr>
<th></th>
<th>Fixed-Point</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost/Unit</td>
<td>$5 - $79</td>
<td>$5 - $381</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>Accumulator</td>
<td>load-store or memory-register</td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td>2-4 data, 8 address</td>
<td>8-16 data, 8-16 address</td>
<td></td>
</tr>
<tr>
<td>Data Words</td>
<td>16 or 24 bit</td>
<td>32 bit</td>
<td></td>
</tr>
<tr>
<td>Chip Memory</td>
<td>2-64K data and program</td>
<td>8-64K data and program</td>
<td></td>
</tr>
<tr>
<td>Address Space</td>
<td>16-128K data, 16-64K program</td>
<td>16M – 4Gdata, 16M – 4G program</td>
<td></td>
</tr>
<tr>
<td>Compilers</td>
<td>Bad C</td>
<td>Better C, C++</td>
<td></td>
</tr>
<tr>
<td>Examples</td>
<td>TI TMS320C5x; Motorola 56000</td>
<td>TI TMS320C3x; Analog Devices SHARC</td>
<td></td>
</tr>
</tbody>
</table>

Conventional DSPs

- Market share: 95% fixed-point, 5% floating-point
- Each processor comes in dozens of configurations
  - Data and program memory size
  - Peripherals: A/D, D/A, serial, parallel ports, timers
- Drawbacks
  - No byte addressing (needed for image and video)
  - Limited on-chip memory
  - Limited addressable memory on most fixed-point DSPs
  - Non-standard C extensions to support fixed-point data
DSP Example

- Finite Impulse Response filter (FIR)
- Can be used for lowpass, highpass, bandpass, etc.
- Basic DSP operation

For each sample, computes

\[ y_n = \sum_{i=0}^{k} a_i x_{n+i} \]

- \( a_0 \ldots a_k \) are filter coefficients
- \( x_n \) and \( y_n \) are the nth input and output sample

56001 Datapath

56001 Memory Spaces

- Three memory regions, each 64K:
  - 24-bit Program memory
  - 24-bit X data memory
  - 24-bit Y data memory

- Idea: enable simultaneous access of program, sample, and coefficient memory
- Three on-chip memory spaces can be used this way
- One off-chip memory pathway connected to all three memory spaces
- Only one off-chip access per cycle maximum

56001 Address Generation

- Addresses come from pointer register \( r0 \ldots r7 \)
- Offset registers \( n0 \ldots n7 \) can be added to pointer
- Modifier registers cause the address to wrap around
- Zero modifier causes reverse-carry arithmetic

<table>
<thead>
<tr>
<th>Address Notation</th>
<th>Next value of r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0 + n0</td>
<td>(r0) + n0</td>
</tr>
<tr>
<td>r0 + n0</td>
<td>(r0) + n0</td>
</tr>
<tr>
<td>r0 + n0</td>
<td>(r0) + n0</td>
</tr>
<tr>
<td>r0 – 1</td>
<td>–(r0)</td>
</tr>
<tr>
<td>r0 – 1</td>
<td>–(r0)</td>
</tr>
<tr>
<td>r0 – 1</td>
<td>–(r0)</td>
</tr>
<tr>
<td>r0 – 1</td>
<td>–(r0)</td>
</tr>
<tr>
<td>r0 – 1</td>
<td>–(r0)</td>
</tr>
<tr>
<td>r0 – 1</td>
<td>–(r0)</td>
</tr>
<tr>
<td>r0 – 1</td>
<td>–(r0)</td>
</tr>
</tbody>
</table>

FIR Filter in 56001

Define symbolic constants

- \( n \) equ 20
- \( \text{start} \) equ $40
- \( \text{samples} \) equ $0
- \( \text{coefficients} \) equ $0
- \( \text{output} \) equ $ffe1

Addresses of memory-mapped I/O

- \( \text{org p:start} \)
- \( \text{move #samples, r0} \)
- \( \text{move #coefficients, r4} \)
- \( \text{move #n-1, m0} \)
- \( \text{move m0, m4} \)

“Locate this in program memory at $40”

“Initialize pointers to samples and coefficients”

“Prepare to treat these as circular buffers of size n”
**FIR Filter in 56001**

```
movep y:input, x: (r0)
clr a x: (r0)+, x0 y: (r4)+, y0
rep #n-1
mac x0,y0,a x: (r0)+, x0 y: (r4)+, y0
macr x0,y0,a (r0)-
movp a, y: output
```

"Load a sample from an I/O device in Y data memory"

"Clear accumulator A"

"Load a sample from X memory into x0, advance the pointer"

"Load a coefficient from Y memory into y0, advance the pointer"

"Repeat the next instruction n-1 times"

"Fetch next sample and coefficient"

"a = a + x0 * y0"

"Write the filtered result to an I/O device in Y data memory"

---

**FIR Filter in 56001**

```
movep y:input, x: (r0)
clr a x: (r0)+, x0 y: (r4)+, y0
```

"Repeat the next instruction n-1 times"

```
mac x0,y0,a x: (r0)+, x0 y: (r4)+, y0
macr x0,y0,a (r0)-
movp a, y: output
```

---

**TI TMS320C6000 VLIW DSP**

- Eight instruction units dispatched by one very long instruction word
- Designed for DSP applications
- Orthogonal instruction set
- Big, uniform register file (16 32-bit registers)
- Better compiler target than 56001
- Deeply pipelined (up to 15 levels)
- Complicated, but more regular, datapath

---

**Pipelining on the C6**

- One instruction issued per clock cycle
- Very deep pipeline
  - 4 fetch cycles
  - 2 decode cycles
  - 1-10 execute cycles
- Branch in pipeline disables interrupts
- Conditional instructions avoid branch-induced stalls
- No hardware to protect against hazards
  - Assembler or compiler's responsibility

---

**'C6 Datapath**
'C6 Datapath

- Two identical halves
- Each has
  - 16 32-bit registers
  - Logical/Arithmetic (.L)
  - Shifter/Branching (.S)
  - Multiplier (.M)
  - Data/Memory (.D)
- One cross path

FIR in 'C6 Assembly

"Load a halfword (16 bits)"
FIRLOOP:
|| LDH .D1 *A1++, A2 ; Fetch next sample
|| LDH .D2 *B1++, B2 ; Fetch next coefficient
|| [B0] SUB .L2 B0, 1, B0 ; Decrement loop count
|| [B0] B .S2 FIRLOOP ; Branch if non-zero
|| MPY .M1X A2, B2, A3 ; Sample * Coefficient
|| ADD .L1 A4, A3, A4 ; Accumulate result
X: "Use the cross path"
predicated instruction:
"Execute only if B0 is non-zero"
"Run all of these instructions in parallel"

Peripherals

- Often the whole point of the system
- Memory-mapped I/O
  - Magical memory locations that make something happen or change on their own
- Typical meanings:
  - Configuration (write)
  - Status (read)
  - Address/Data (access more peripheral state)

Example: 56001 Port C

- Nine pins each usable in one of two ways
  - Simple parallel I/O
  - Serial interface

  Parallel Serial
  --- ---
  PC0 RxD
  PC1 TxD
  PC2 SCLK
  Serial Communication Interface (SCI)

  PC3 SC0
  PC4 SC1
  PC5 SC2
  PC6 SCK
  PC7 SRD
  PC8 STD
  Synchronous Serial Interface (SSI)

Port C Registers for Parallel Port

- Port C Control Register
  - Selects mode (parallel or serial) of each pin
  X: $FFE1

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>parallel I/O</td>
</tr>
<tr>
<td>1</td>
<td>serial I/O</td>
</tr>
</tbody>
</table>

- Port C Data Direction Register
  - I/O direction when used in parallel mode
  X: $FFE3

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Input</td>
</tr>
<tr>
<td>1</td>
<td>Output</td>
</tr>
</tbody>
</table>

Port C Registers for Parallel Port

- Port C Data Register
  - Returns input data or sets output state of parallel port
  X: $FFE5

  Read: pin state
  Write: set output pin state
Port C SCI

- Three-pin interface
- 422 Kbit/s NRZ asynchronous interface (RS-232-like)
- 3.375 Mbit/s synchronous serial mode
- Multidrop mode for multiprocessor systems
- Two Wakeup modes
  - Idle line
  - Address bit
- Wired-OR mode
- On-chip or external baud rate generator
- Four interrupt priority levels

Port C SCI Registers

- X: $FF0
- SCI Control Register
- Word select bits
- Shift direction
- Send break
- Wakeup mode select
- Receiver wakeup enable
- Wired-OR mode select
- Receiver Enable
- Transmitter Enable
- Idle line interrupt enable
- Receive interrupt enable
- Transmit interrupt enable
- Timer interrupt enable
- Clock polarity

Port C SCI Registers

- X: $FF1
- SCI Status Register (read-only)
- Transmitter Empty
- Transmitter Reg Empty
- Receive Data Full
- Idle Line
- Overrun Error
- Parity Error
- Framing Error
- Received bit 8

Port C SSI

- Intended for synchronous, constant-rate protocols
  - Easy interface to serial ADCs and DACs
- Many more operating modes than SCI
- Six Pins (Rx, Tx, Clk, Rx Clk, Frame Sync, Tx Clk)
- 8, 12, 16, or 24-bit words

Port C SSI Registers

- $FFEC SSI Control Register A
  - prescaler, frame rate, word length
- $FFED SSI Control Register B
  - Interrupt enables, various mode settings
- $FFEE SSI Status/Time Slot Register
  - Sync, empty, overrun
- $FFEF SSI Receive/Transmit Data Register