Chapter 3  General-Purpose Processors: Software

Introduction

- General-Purpose Processor
  - Processor designed for a variety of computation tasks
  - Low unit cost, in part because manufacturer spreads NRE over large numbers of units
    - Motorola sold half a billion 68HC05 microcontrollers in 1996 alone
  - Carefully designed since higher NRE is acceptable
    - Can yield good performance, size and power
  - Low NRE cost, short time-to-market/prototype, high flexibility
    - User just writes software; no processor design
  - a.k.a. "microprocessor" – ‘micro' used when they were implemented on one or a few chips rather than entire rooms

Basic Architecture

- Control unit and datapath
  - Note similarity to single-purpose processor
- Key differences
  - Datapath is general
  - Control unit doesn’t store the algorithm – the algorithm is “programmed” into the memory
Datapath Operations

- **Load**
  - Read memory location into register
- **ALU operation**
  - Input certain registers through ALU, store back in register
- **Store**
  - Write register to memory location

Control Unit

- Control unit configures the datapath operations
  - Sequence of desired operations ("instructions") stored in memory - "program"
  - Instruction cycle - broken into several sub-operations, each one clock cycle, e.g.:
    - Fetch: Get next instruction into IR
    - Decode: Determine what the instruction means
    - Fetch operands: Move data from memory to datapath register
    - Execute: Move data through the ALU
    - Store results: Write data from register to memory

Control Unit Sub-Operations

- **Fetch**
  - Get next instruction into IR
  - PC: program counter, always points to next instruction
  - IR: holds the fetched instruction
Control Unit Sub-Operations

- **Decode**
  - Determine what the instruction means

- **Fetch operands**
  - Move data from memory to datapath register

- **Execute**
  - Move data through the ALU
  - This particular instruction does nothing during this sub-operation
Control Unit Sub-Operations

- **Store results**
  - Write data from register to memory
  - This particular instruction does nothing during this sub-operation

Instruction Cycles

- **PC=100**
  - Fetch
  - Decode
  - Ops: Add, Sub, etc.
  - Exac: Store results

- **PC=101**
  - Fetch
  - Decode
  - Ops: Add, Sub, etc.
  - Exac: Store results

- **PC=102**
  - R0
  - R1
  - Load R0, M[500]
  - Inc R1, R0
  - Store M[501], R1
Instruction Cycles

- PC=100
  Fetch Decode Exec. Store
- PC=101
  Fetch Decode Exec. Store
- PC=102
  Fetch Decode Exec. Store

Architectural Considerations

- N-bit processor
  - N-bit ALU, registers, buses, memory data interface
  - Embedded: 8-bit, 16-bit, 32-bit common
  - Desktop/servers: 32-bit, even 64
- PC size determines address space

Architectural Considerations

- Clock frequency
  - Inverse of clock period
  - Must be longer than longest register to register delay in entire processor
  - Memory access is often the longest
Pipelining: Increasing Instruction Throughput

Superscalar and VLIW Architectures

- Performance can be improved by:
  - Faster clock (but there’s a limit)
  - Pipelining: slice up instruction into stages, overlap stages
  - Multiple ALUs to support more than one instruction stream
    - Superscalar
      - Scalar: non-vector operations
      - Fetches instructions in batches, executes as many as possible
      - May require extensive hardware to detect independent instructions
    - VLIW: each word in memory has multiple independent instructions
      - Relies on the compiler to detect and schedule instructions
      - Currently growing in popularity

Two Memory Architectures

- Princeton
  - Fewer memory wires
- Harvard
  - Simultaneous program and data memory access
Cache Memory

- Memory access may be slow
- Cache is small but fast memory close to processor
  - Holds copy of part of memory
  - Hits and misses

Processor

Cache

Memory

Processor Cache Memory

Fast/expensive technology, usually on the same chip

Slower/cheaper technology, usually on a different chip

Programmer’s View

- Programmer doesn’t need detailed understanding of architecture
  - Instead, needs to know what instructions can be executed
- Two levels of instructions:
  - Assembly level
  - Structured languages (C, C++, Java, etc.)
- Most development today done using structured languages
  - But, some assembly level programming may still be necessary
  - Drivers: portion of program that communicates with and/or controls (drives) another device
    - Often have detailed timing considerations, extensive bit manipulation
    - Assembly level may be best for these

Assembly-Level Instructions

- Instruction Set
  - Defines the legal set of instructions for that processor
    - Data transfer: memory/register, register/register, I/O, etc.
    - Arithmetic/logical: move register through ALU and back
    - Branches: determine next PC value when not just PC+1
Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Operand Field</th>
<th>Register file contents</th>
<th>Memory contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>Data</td>
<td>Register address</td>
<td>Data</td>
</tr>
<tr>
<td>Register-direct</td>
<td>Register address</td>
<td>Memory address</td>
<td>Memory address</td>
</tr>
<tr>
<td>Register indirect</td>
<td>Register indirect</td>
<td>Memory address</td>
<td>Memory address</td>
</tr>
<tr>
<td>Direct</td>
<td>Memory address</td>
<td>Memory address</td>
<td>Data</td>
</tr>
<tr>
<td>Indirect</td>
<td>Memory address</td>
<td>Memory address</td>
<td>Memory address</td>
</tr>
</tbody>
</table>

Programmer Considerations

- Program and data memory space
  - Embedded processors often very limited
    - e.g., 64 Kbytes program, 256 bytes of RAM (expandable)
- Registers: How many are there?
  - Only a direct concern for assembly-level programmers
- I/O
  - How communicate with external signals?
- Interrupts

Development Environment

- Development processor
  - The processor on which we write and debug our programs
  - Usually a PC
- Target processor
  - The processor that the program will run on in our embedded system
  - Often different from the development processor
**Software Development Process**

- Compilers
  - Cross compiler
    - Runs on one processor, but generates code for another
- Assemblers
- Linkers
- Debuggers
- Profilers

**Running a Program**

- If development processor is different than target, how can we run our compiled code? Two options:
  - Download to target processor
  - Simulate
- Simulation
  - One method: Hardware description language
    - But slow, not always available
  - Another method: Instruction set simulator (ISS)
    - Runs on development processor, but executes instructions of target processor

**Instruction Set Simulator For A Simple Processor**
Testing and Debugging

- ISS
  - Gives us control over time – set breakpoints, look at register values, set values, step-by-step execution, ...
  - But, doesn’t interact with real environment
- Download to board
  - Use device programmer
  - Runs in real environment, but not controllable
- Compromise: emulator
  - Runs in real environment, at speed or near
  - Supports some controllability from the PC

Application-Specific Instruction-Set Processors (ASIPs)

- General-purpose processors
  - Sometimes too general to be effective in demanding application
    - e.g., video processing – requires huge video buffers and operations on large arrays of data, inefficient on a GPP
    - But single-purpose processor has high NRE, not programmable
- ASIPs – targeted to a particular domain
  - Contain architectural features specific to that domain
    - e.g., embedded control, digital signal processing, video processing, network processing, telecommunications, etc.
  - Still programmable

A Common ASIP: Microcontroller

- For embedded control applications
  - Reading sensors, setting actuators
  - Mostly dealing with events (bits): data is present, but not in huge amounts
    - e.g., VCR, disk drive, digital camera (assuming SPP for image compression), washing machine, microwave oven
- Microcontroller features
  - On-chip peripherals
    - Timers, analog-digital converters, serial communication, etc.
  - Tightly integrated for programmer, typically part of register space
  - On-chip program and data memory
  - Direct programmer access to many of the chip’s pins
  - Specialized instructions for bit-manipulation and other low-level operations
Another Common ASIP: Digital Signal Processors (DSP)

- For signal processing applications
  - Large amounts of digitized data, often streaming
  - Data transformations must be applied fast
  - e.g., cell-phone voice filter, digital TV, music synthesizer

- DSP features
  - Several instruction execution units
  - Multiple-accumulate single-cycle instruction, other instrs.
  - Efficient vector operations – e.g., add two arrays
    - Vector ALUs, loop buffers, etc.

Trend: Even More Customized ASIPs

- In the past, microprocessors were acquired as chips
- Today, we increasingly acquire a processor as Intellectual Property (IP)
  - e.g., synthesizable VHDL model
- Opportunity to add a custom datapath hardware and a few custom instructions, or delete a few instructions
  - Can have significant performance, power and size impacts
  - Problem: need compiler/debugger for customized ASIP
    - Remember, most development uses structured languages
    - One solution: automatic compiler/debugger generation
      - e.g., www.tensillica.com
    - Another solution: retargettable compilers
      - e.g., www.improvsys.com (customized VLIW architectures)

Selecting a Microprocessor

- Issues
  - Technical: speed, power, size, cost
  - Other: development environment, prior expertise, licensing, etc.
- Speed: how evaluate a processor’s speed?
  - Clock speed – but instructions per cycle may differ
  - Instructions per second – but work per instr. may differ
    - MIPS: 1 MIPS = 1,757 Dhrystones per second (based on Digital’s VAX 11/780). A.k.a. Dhrystone MIPS. Commonly used today.
      - So, 750 MIPS = 750*1,757 = 1,317,750 Dhrystones per second
    - SPEC: set of more realistic benchmarks, but oriented to desktops
        - Suites of benchmarks: automotive, consumer electronics, networking, office automation, telecommunications
Chapter Summary

- General-purpose processors
  - Good performance, low NRE, flexible
- Controller, datapath, and memory
- Structured languages prevail
  - But some assembly level programming still necessary
- Many tools available
  - Including instruction-set simulators, and in-circuit emulators
- ASIPs
  - Microcontrollers, DSPs, network processors, more customized ASIPs
- Choosing among processors is an important step
- Designing a general-purpose processor is conceptually the same as designing a single-purpose processor