SESSION 9: INSTRUCTION ARCHITECTURE

Reading: Section 4.14 (Intro); Chapter 5, except stack architecture details (page 275-280) and expanding op-codes (Sec.5.2.5)

Objectives

• Obtain a more detailed look at different instruction formats, operand types, and memory access methods
• See the interrelation between machine organization and instruction formats
• Understand the difference between CISC and RISC architectures
• Understand memory addressing modes
• Understand instruction-level pipelining and its affect upon execution performance
Instruction Formats

• Instruction sets are differentiated by:
  • Number of bits per instruction
  • Stack-based or register-based architecture
  • Number and category of operations
  • Operand features

MARIE only has a simple operand format

Register-based architectures dominate today

Instruction Formats

• Instruction set architectures are measured by:
  • Main memory space occupied by a program
  • Instruction complexity
  • Instruction length (in bits)
  • Total number of instructions in the instruction set
Instruction Formats

• Instruction set considerations include:
  • Instruction length
    • Short, long, or variable
  • Number of operands
  • Type, location, and size of operands
  • Number of addressable registers
  • Memory organization
    • Byte or word addressable
  • Addressing modes
    • Examples: direct, indirect, and indirect + offset

Instruction Formats

• Byte ordering is another major architectural consideration
  • If we have a two-byte integer, the integer may be stored so that the least significant byte is followed by the most significant byte or vice versa
    • In little endian machines, the least significant byte is followed by the most significant byte
    • Big endian machines store the most significant byte first (at the lower address)

Terms are from Gulliver’s Travels
Endian Formats

• As an example, suppose we have the hexadecimal number 12345678.
• The big endian and small endian arrangements of the bytes are shown below.

<table>
<thead>
<tr>
<th>Address</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Endian</td>
<td>12</td>
<td>34</td>
<td>56</td>
<td>78</td>
</tr>
<tr>
<td>Little Endian</td>
<td>78</td>
<td>56</td>
<td>34</td>
<td>12</td>
</tr>
</tbody>
</table>

Big Endian Vs. Little Endian

• Big endian advantages:
  • Is more natural (more understandable hex dumps)
  • The sign of the number can be determined by looking at the byte at address offset 0
  • Strings and integers are stored in the same order
• Little endian advantages:
  • Easier to place values on non-word boundaries
  • Conversion from a 16-bit integer address to a 32-bit integer address does not require any arithmetic.
Instruction Formats

• Architecture design choices concern how the CPU will store data
  • Stack architecture – somewhat dated, although most computers now use a memory stack
  • Accumulator architecture – with accumulator architectures (e.g., MARIE), an implicit operand is the accumulator
  • General purpose register architecture.

• Tradeoffs
  • Simplicity (and cost) of hardware design
  • Execution speed and ease of use

GPR Systems

• Most systems today are GPR (General Purpose Register) systems, either
  • Memory-memory - two or three operands may be in memory
  • Register-memory (e.g., Intel) - at least one operand must be in a register
  • Load-store - no operands may be in memory (data is moved into registers before operations are performed)

• The number of operands and the number of available registers have a direct affect on instruction length
Operands and Instruction Length

- Maximum number of operands affects instruction length
- Instruction formatting
  - Fixed length wastes space, but fast
  - Variable length – complex to decode, but saves space
- Typical modern computers use 2-3 different instruction lengths
- Instructions need to be word aligned

Instruction Formats

- We have seen how instruction length is affected by the number of operands supported by the ISA
- In any instruction set, not all instructions require the same number of operands
- Operations that require no operands (e.g., HALT) waste some space for fixed-length ISAs

MARIE instructions have 0 or 1 operands

We do not cover expanding opcodes in detail

All architectures have some limit on the number of operands
Are We on Track?

- A computer has 32-bit instructions and 12-bit addresses. Suppose there are 250 two-address instructions. How many one-address instructions can be formulated?

<table>
<thead>
<tr>
<th>Op code (8 bits)</th>
<th>Address 1 (12 bits)</th>
<th>Address 2 (12 bits)</th>
</tr>
</thead>
</table>

8 bit op code allows for 256 operations (250 two address and 6 one-address)

Instruction Types

- Instructions fall into several broad categories:
  - Data movement
  - Arithmetic
  - Boolean
  - Bit manipulation (including shift and rotate)
  - I/O
  - Control transfer
  - Special purpose (e.g., string processing, protection, cache management, etc.)

What are some high-level language examples of each?

Consistency and orthogonality
Addressing Issues

• Data Types
  • Integer, floating point, pointers, character, etc.
  • Range, length
• Address Modes - specifies where an operand is located
  • MARIE allowed either a memory address or a pointer to a memory address
  • The actual location of an operand is its effective address

Addressing Modes …

• Immediate addressing - data is part of the instruction
  • e.g., LOADIMMEDIATE 008 loads the numeric value 8 into the AC
• Direct addressing - address of the data is given in the instruction
  • e.g., LOAD 008 with direct addressing loads the value stored in memory location 8 into the AC
• Register addressing - data is located in a register
  • Same as direct addressing, except the data is located in a register (not memory)
… Addressing Modes

• Indirect addressing gives the address of the address of the data in the instruction
  • In register indirect addressing a register is used instead of a memory location
• Indexed addressing uses a register (implicitly or explicitly) as an offset
  • Offset is added to the address in the operand to determine the effective address of the data
• Based addressing is similar except that a base register is used instead of an index register.

Address is Base + Offset (index)

Based addressing is useful in sequentially accessing arrays and strings

Based Addressing

Base register contains the address of the start of the array

Index register contains offset (e.g., 3)

Calculated address is sum of base register plus index register
Why are There So Many Addressing Modes?

- Performance
- Program relocation
- Very large memories

Instruction Pipelining

- Some CPUs divide the fetch-decode-execute cycle into smaller steps
- These smaller steps might be executed in parallel to increase performance
- Such parallel execution is called instruction pipelining
- Instruction pipelining provides for instruction level parallelism (ILP)

Like a car wash
Speedup

- **Speedup** is a metric for relative performance improvement when executing a task
- Speedup = $T_{\text{old}} / T_{\text{new}}$
  - $T_{\text{old}}$ – old execution time
  - $T_{\text{new}}$ – new execution time

Instruction Pipelining Example

- Let’s break a fetch-decode-execute cycle into smaller steps and
- Suppose we have a six-stage pipeline
  - S1 fetches the instruction
  - S2 decodes it
  - S3 determines the address of the operands
  - S4 fetches operands
  - S5 executes the instruction
  - S6 stores the result

Remember the microcode examples
Instruction Pipelining

- For every clock cycle, one small step is carried out, and the stages are overlapped.

To avoid stalling the pipeline, execution cycles should be consistent.

Instruction Pipelining

- An instruction pipeline may stall, or be flushed for any of the following reasons:
  - Resource conflicts
  - Data dependencies
  - Conditional branching

- Measures can be taken in software and hardware to reduce the effects of these hazards

Processors use many advanced features to keep the pipeline full.

Some computers offer multiple processor components to create “superscalar” performance.

Compilers can also be used to improve pipeline performance.
Are We on Track?

• A non-pipelined system takes 200ns to process a task. The same task can be processed in a 5-segment pipeline with a clock cycle of 40ns
• For 200 tasks, calculate:
  • \( T_{old} \)
  • \( T_{new} \)
  • Speedup
  • Maximum speedup (assume an infinite number of tasks)

\[ T_{old} = 200 \text{ns} \times 200 \text{ tasks} = 40,000 \text{ns} = 40 \text{ microseconds} \]

We calculate \( T_{new} \) by noting that each task starts 40ns after the previous task.
Therefore, the last task completes the first step in \( 200 \times 40 \text{ns} = 8,000 \text{ns} = 8 \text{ microseconds} \)
The last step needs 4 more steps to complete so \( T_{new} = 8,000 \text{ns} + 160 \text{ns} = 8,160 \text{ns} \)

\[ \text{Speedup} = \frac{40,000}{8,160} = 4.90 \]
\[ \text{Maximum speedup} = 5 \]
Real World Architectures

- MARIE shares many features with modern architectures but MARIE is much simpler
- Real-world architecture categories.
  - CISC (complex instruction set computer)
  - RISC (reduced instruction set computer)
- Current CPU Architectures
  - Intel x86 family (including AMD)
  - ARM (mobile devices)
  - IBM Mainframe series

Currently, there are a limited number of dominant computer architectures

CISC

- Early processor architectures attempted to include instructions compatible with high level languages
- Features
  - Large instruction set
  - More complex implementation
  - Smaller program size
  - Fewer memory accesses
Instruction Level Parallelism

- Measure of how many computer operations can be performed simultaneously
- Typically thought of as a measure of parallelism in a non-parallel program
- Implemented by compilers and microcode
- Techniques (explored later in the course)
  - Instruction pipelining
  - Multiple execution units
  - Out-of-order execution
  - Register renaming - avoids serialization due to register reuse
  - Speculative execution - execution of instructions before being certain whether they’re necessary
  - Branch prediction

RISC

- Simplified instruction set
- Allowed more instruction parallelism
- Initial RISC processors demonstrated performance advantages over then-current CISC processors

Boundary between RISC and CISC is no longer as pronounced
RISC Machines …

- **RISC**
  - Simple instructions, few in number
  - Fixed length instructions
  - Complexity in compiler
  - Only LOAD/STORE instructions access memory
  - Few addressing modes

- **CISC**
  - Many complex instructions
  - Variable length instructions
  - Complexity in microcode
  - Many instructions can access memory
  - Many addressing modes

… RISC Machines

- **RISC**
  - Multiple register sets
  - 3 operands per instruction
  - Single-cycle instructions
  - Hardwired control
  - Highly pipelined

- **CISC**
  - Single register set
  - One or two register operands per instruction
  - Multiple cycle instructions
  - Microprogrammed control
  - Less pipelined
Current CPU Architectures

- Current CPU Architectures
  - Intel x86 family (including AMD)
  - ARM (mobile devices)
  - IBM Mainframe series

Have You Met the Objectives?

- Obtain a more detailed look at different instruction formats, operand types, and memory access methods
- See the interrelation between machine organization and instruction formats
- Understand memory addressing modes
- Understand instruction-level pipelining and its affect upon execution performance