ADDRESSING MODES AND PIPELINING

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Tonight: Homework 1 release (due on Sept. 4th)
    - Verify your uploaded files before deadline

- This lecture
  - RISC vs. CISC
  - Addressing modes
  - Pipelining
ISA Types

Operand locations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Stack</th>
<th>Accumulator</th>
<th>Register-memory</th>
<th>Register-register/Load-store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push A</td>
<td>Load A</td>
<td>Load R1,A</td>
<td>Load R1,A</td>
<td></td>
</tr>
<tr>
<td>Push B</td>
<td>Add B</td>
<td>Add R3,R1,B</td>
<td>Load R2,B</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Store C</td>
<td>Store R3,C</td>
<td>Add R3,R1,R2</td>
<td>Store R3,C</td>
</tr>
<tr>
<td>Pop C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Which Set of Instructions?

- ISA influences the execution time
  - CPU time = IC x CPI x CT
- Complex Instruction Set Computing (CISC)

- Reduced Instruction Set Computing (RISC)
Which Set of Instructions?

- ISA influences the execution time
  - CPU time = IC x CPI x CT

- Complex Instruction Set Computing (CISC)
  - May reduce IC, increase CPI, and increase CT
  - CPU time may be increased

- Reduced Instruction Set Computing (RISC)
  - May increases IC, reduce CPI, and reduce CT
  - CPU time may be decreased
RISC vs. SISC

RISC ISA

- Simple operations
  - Simple and fast FU
- Fixed length
  - Simple decoder
- Limited inst. formats
  - Easy code generation

CISC ISA

- Complex operations
  - Costly memory access
- Variable length
  - Complex decoder
- Limited registers
  - Hard code generation
Memory Addressing

- Register
  - Add r4, r3
- Immediate
  - Add r4, #3
- Displacement
  - Add r4, 100(r1)
- Register indirect
  - Add r4, (r1)
Memory Addressing

- **Register**
  - Add r4, r3 \( \text{Reg}[4] = \text{Reg}[4] + \text{Reg}[3] \)

- **Immediate**
  - Add r4, #3 \( \text{Reg}[4] = \text{Reg}[4] + 3 \)

- **Displacement**
  - Add r4, 100(r1) \( \ldots + \text{Mem}[100 + \text{Reg}[1]] \)

- **Register indirect**
  - Add r4, (r1) \( \ldots + \text{Mem}[	ext{Reg}[1]] \)
Memory Addressing

- Indexed
  - Add r3, (r1+r2)

- Direct
  - Add r1, (1001)

- Memory indirect
  - Add r1, @(r3)

- Auto-increment
  - Add r1, (r2)
Memory Addressing

- Indexed
  - Add r3, (r1+r2) ... + Mem[Reg[1] + Reg[2]]

- Direct
  - Add r1, (1001) ... + Mem[1001]

- Memory indirect
  - Add r1, @(r3) ... + Mem[Mem[Reg[3]]]

- Auto-increment
  - Add r1, (r2) + ... + Mem[Reg[2]]
Memory Addressing

- Auto-decrement
  - Add r1, -(r2)

- Scaled
  - Add r1, 100(r2)[r3]
Memory Addressing

- **Auto-decrement**
  - Add r1, -(r2) \( \Rightarrow \text{Reg}[2] = \text{Reg}[2] - d \)
  - \( \ldots + \text{Mem}[\text{Reg}[2]] \)

- **Scaled**
  - Add r1, 100(r2)[r3]
  - \( \ldots + \text{Mem}[100 + \text{Reg}[2] + \text{Reg}[3] \times d] \)
Example Problem

- Find the effective memory address
  - Add r2, 200(r1)
  - Add r2, (r1)
  - Add r2, @(r1)

<table>
<thead>
<tr>
<th>Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>100</td>
</tr>
<tr>
<td>r2</td>
<td>200</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
</tr>
<tr>
<td>500</td>
<td>800</td>
</tr>
</tbody>
</table>
Example Problem

- Find the effective memory address
  - Add r2, 200(r1)
    - \( r2 = r2 + \text{Mem}[300] \)
  - Add r2, (r1)
    - \( r2 = r2 + \text{Mem}[100] \)
  - Add r2, @(r1)
    - \( r2 = r2 + \text{Mem}[400] \)
A guideline for generating/interpreting instructions

Example: MIPS
- Fixed size 32-bit instructions
- Three opcode types
  - I-type: load, store, conditional branch
  - R-type: ALU operations
  - J-type: jump
Pipelining
Every RISC instruction may require multiple processing steps
Every RISC instruction may require multiple processing steps.
Every RISC instruction may require multiple processing steps:
- Instruction Fetch (IF)
- Instruction Decode (ID)
- Register Read (RR)
  - All instructions?
- Execute Instructions (EXE)
- Memory Access (MEM)
  - All instructions?
- Register Write Back (WB)
Single-cycle RISC Architecture

- Example: simple MIPS architecture
  - Critical path includes all of the processing steps

![Diagram of a single-cycle RISC architecture with components such as Controller, PC, Inst. Fetch, Inst. Memory, Inst. Decode, Register File, ALU, Execute, Memory, Data Memory, and Write Back connections.]
Single-cycle RISC Architecture

Example program

CT=6ns; CPU Time = ?

AND R1, R2, R3
XOR R4, R2, R3
SUB R5, R1, R4
ADD R6, R1, R4
MUL R7, R5, R6

CPU Time = CI x CPI x CT
Example program

CT=6ns; CPU Time = 5 x 1 x 6ns = 30ns

AND R1,R2,R3
XOR R4,R2,R3
SUB R5,R1,R4
ADD R6,R1,R4
MUL R7,R5,R6

How to improve?

CPU Time = CI x CPI x CT
Reusing Idle Resources

- Each processing step finishes in a fraction of a cycle
  - Idle resources can be reused for processing next instructions
Pipelined Architecture

- Five stage pipeline
- Critical path determines the cycle time

![Diagram of a five-stage pipeline with timing values: 1.5ns, 1.05ns, 1.25ns, and 1.5ns for each stage.](image-url)
Pipelined Architecture

Example program

CT = 1.5ns; CPU Time = ?

AND R1,R2,R3

XOR R4,R2,R3

SUB R5,R1,R4

ADD R6,R1,R4

MUL R7,R5,R6

CPU Time = CI x CPI x CT
Pipelined Architecture

- Example program
  - CT=1.5ns; CPU Time = 5 x 5 x 1.5ns = 37.5ns > 30ns

```
AND R1, R2, R3
XOR R4, R2, R3
SUB R5, R1, R4
ADD R6, R1, R4
MUL R7, R5, R6
```

CPU Time = Cl x CPI x CT

Time

WORSE!!
Pipelined Architecture

Example program

CT=1.5ns; CPU Time = ?

AND R1,R2,R3

XOR R4,R2,R3

SUB R5,R1,R4

ADD R6,R1,R4

MUL R7,R5,R6

CPU Time = CI x CPI x CT
Example program

CT = 1.5ns; CPU Time = 5 x 1 x 1.5ns = 7.5ns

AND R1, R2, R3

XOR R4, R2, R3

SUB R5, R1, R4

ADD R6, R1, R4

MUL R7, R5, R6

What is the cost of pipelining?

CPU Time = Cl x CPI x CT
Pipelined Architecture

Example program

- CT=1.5ns; CPU Time = 5 x 1 x 1.5ns = 7.5ns

AND R1,R2,R3
XOR R4,R2,R3
SUB R5,R1,R4
ADD R6,R1,R4
MUL R7,R5,R6

What is the cost of pipelining?

CT: original cycle time
P: no. pipeline stages
\( t \): additional HW delay

New Time = CI \times CPI \times (t + CT/P)
Speedup = \frac{CT}{t + CT/P}

CPU Time = CI \times CPI \times CT
Pipelined Architecture

Example program

CT = 1.5ns; CPU Time = 5 x 1 x 1.5ns = 7.5ns

AND R1, R2, R3

XOR R4, R2, R3

SUB R5, R1, R4

ADD R6, R1, R4

MUL R7, R5, R6

What is the cost of pipelining?

CT: original cycle time
P: no. pipeline stages
t: additional HW delay
New Time = CI x CPI x (t + CT/P)
Speedup = CT / (t + CT/P)

What about CPI?

CPU Time = CI x CPI x CT