Processor Architecture

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Introduction (2)

<table>
<thead>
<tr>
<th>Cray® X-MP</th>
<th>Intel Pentium® 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year:</td>
<td>1982</td>
</tr>
<tr>
<td>Freq.</td>
<td>105MHz</td>
</tr>
<tr>
<td>Perp.</td>
<td>400M FLOPS</td>
</tr>
<tr>
<td>Power</td>
<td>? KW</td>
</tr>
<tr>
<td>Size</td>
<td>2m</td>
</tr>
<tr>
<td>Weight</td>
<td>&gt;100kg</td>
</tr>
<tr>
<td>Cost</td>
<td>$15M</td>
</tr>
<tr>
<td>Cooling</td>
<td>Freon</td>
</tr>
<tr>
<td></td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>2000MHz</td>
</tr>
<tr>
<td></td>
<td>2G FLOPS</td>
</tr>
<tr>
<td></td>
<td>100W</td>
</tr>
<tr>
<td></td>
<td>2 cm²</td>
</tr>
<tr>
<td></td>
<td>&lt;1g</td>
</tr>
<tr>
<td></td>
<td>&lt;500$</td>
</tr>
<tr>
<td></td>
<td>Air</td>
</tr>
</tbody>
</table>

* The Cray® X-MP system sits in UPC, Barcelona

Source: Intel
Trends (1)

- Trends in computer architectures
  - Pre-WWII: Mechanical calculating machines
  - WWII-50’s: Technology improvement
    - Relays → vacuum tubes
    - High-level languages
  - 60’s: Miniaturization/Packaging
    - Transistors
    - Integrated Circuits (ICs)
  - 70’s: Semantic Gap
    - Complex instruction set
    - Large support in hardware
    - Microcoding
Trends (2)

- **Trends in computer architectures (cont’d)**
  - **80’s**: Keep it simple
    - RISC (Reduced Instruction Set Computer)
    - Shift complexity to software
  - **90’s**: What to do with all these transistors?
    - Large on-chip cache
    - Prefetching hardware
    - Speculative execution
    - Special-purpose instructions and hardware
    - Multiple processors on-a-chip
    - ...
Instruction Set Architecture (1)

- **CISC (Complex Instruction Set Computer)**
  - Dominant style through mid-80’s
  - Add instructions to perform “typical” programming tasks
  - Stack-oriented instruction set
    - Use stack to pass arguments, save program counter
    - Explicit push and pop instructions
  - Arithmetic instructions can access memory
    - \texttt{addl \%eax, 12(\%ebx, \%ecx, 4)}
    - Requires memory read/write & complex address calculation
  - Condition codes
    - Set as side effect of arithmetic and logical instructions
Instruction Set Architecture (2)

- **RISC (Reduced Instruction Set Computer)**
  - Fewer, simpler instructions
    - Might take more to get given task done
    - Can be decoded easily
    - Can execute them with small and fast hardware
  - Register-oriented instruction set
    - Many more (typically 32+) registers
    - Use for arguments, return pointer, temporaries
  - Only load and store instructions can access memory
    - Single address mode: base register + displacement
  - No condition codes
    - Test instructions return 0/1 in register
### MIPS Instruction Formats

<table>
<thead>
<tr>
<th>R-type</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>I-type</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit address</th>
</tr>
</thead>
<tbody>
<tr>
<td>beqz</td>
<td>$3, dest</td>
<td>; if (R[3] == 0) PC = PC + 4 + dest</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>J-type</th>
<th>op</th>
<th>26 bit address</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>dest</td>
<td>; PC = dest</td>
</tr>
<tr>
<td>jal</td>
<td>dest</td>
<td>; R[31] = PC + 4, PC = dest</td>
</tr>
</tbody>
</table>
# MIPS Registers

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$zero</td>
<td>The constant value 0</td>
</tr>
<tr>
<td>1</td>
<td>$at</td>
<td>Assembler temporary</td>
</tr>
<tr>
<td>2</td>
<td>$v0</td>
<td>Values for results and expression evaluation</td>
</tr>
<tr>
<td>3</td>
<td>$v1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$a0</td>
<td>Arguments</td>
</tr>
<tr>
<td>5</td>
<td>$a1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$a2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$a3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$t0</td>
<td>Temporaries (Caller-save registers)</td>
</tr>
<tr>
<td>9</td>
<td>$t1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$t2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$t3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>$t4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>$t5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>$t6</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>$t7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>$s0</td>
<td>Saved temporaries (Callee-save registers)</td>
</tr>
<tr>
<td>17</td>
<td>$s1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>$s2</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>$s3</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>$s4</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>$s5</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>$s6</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>$s7</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>$t8</td>
<td>More temporaries (Caller-save registers)</td>
</tr>
<tr>
<td>25</td>
<td>$t9</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>$k0</td>
<td>Reserved for OS kernel</td>
</tr>
<tr>
<td>27</td>
<td>$k1</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>$gp</td>
<td>Global pointer</td>
</tr>
<tr>
<td>29</td>
<td>$sp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>30</td>
<td>$fp</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>31</td>
<td>$ra</td>
<td>Return address</td>
</tr>
</tbody>
</table>
S-MIPS Datapath (5 stages)

- **IF Stage**
- **ID Stage**
- **EX Stage**
- **MEM Stage**
- **WB Stage**

### S-MIPS Datapath Example (1)

![Datapath Diagram](image-url)
S-MIPS Datapath Example (2)

LOAD instruction:  
\[ \text{lw} \; \$r3, \; \$r2, \; 12 \; ; \; \text{R}[3] \leftarrow \text{Mem}[\text{R}[2] + 12] \]
S-MIPS Datapath Example (3)

Branch instruction: `beqz $r3, 16 ; PC += (R[3]==0)? 4+16 : 4`
Pipelining (1)
Pipelining (2)

- **Lessons**
  - Pipelining doesn’t help *latency* of single task, it helps *throughput* of entire workload.
  - Pipeline rate limited by *slowest* pipeline stage.
  - *Multiple* tasks operating simultaneously.
  - Potential speedup = Number pipe stages
  - Unbalanced lengths of pipe stages reduces speedup.
  - Time to “fill” pipeline and time to “drain” it reduces speedup.
Instruction Pipelining

- Why pipelining?
  - Execute billions of instructions, so **throughput** is what matters.

- What is desirable in instruction sets for pipelining?
  - Variable length instructions vs. all instructions same length?
  - Memory operands part of any operation vs. memory operands only in loads or stores?
  - Register operand many places in instruction format vs. registers located in same place?
Pipelined S-MIPS Datapath

IF Stage

ID Stage

EX Stage

MEM Stage

WB Stage

Instr. Memory

Add

+4

PC

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?

MUX

Instr. Memory

Reg File

ALU

Zero?
Visualizing Pipeline

Time (clock cycles)

CC1  CC2  CC3  CC4  CC5  CC6  CC7  CC8  CC9

IM → Reg
IM → Reg
IM → Reg
IM → Reg
IM → Reg
IM → Reg
IM → Reg
IM → Reg
IM → Reg

DM → Reg
DM → Reg
DM → Reg
DM → Reg
DM → Reg
DM → Reg
DM → Reg
DM → Reg
DM → Reg

Reg → DM
Reg → DM
Reg → DM
Reg → DM
Reg → DM
Reg → DM
Reg → DM
Reg → DM
Reg → DM

Filling

Draining

Instruction Order
Pipeline Stall

- Limits to pipelining: hazards prevent the next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support the combination of instructions due to lack of HW capacity
  - Data hazards: Instruction depends on the result of prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow

- Common solution is to stall the pipeline until the hazard is resolved, inserting one or more bubbles (idle clock cycles) in the pipeline.
Structural Hazard

Use dual port memory to support two simultaneous accesses

Access memory from two instructions at the same cycle
Data Hazard

Time (clock cycles)

ADD R1, R2, R3

SUB R4, R1, R3

AND R6, R1, R7

OR R8, R1, R9

XOR R10, R11, R1

Clock Cycle

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

ADD R1, R2, R3

SUB R4, R1, R3

AND R6, R1, R7

OR R8, R1, R9

XOR R10, R11, R1

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri

IM Reg

ALU

DM Reg

A

OR

XOR

Ri

Store into Ri

Read from Ri
Control Hazard

- Three stage stall on branches

10: `beq r1,r3,22`

14: `and r2,r3,r5`

18: `or r6,r1,r7`

22: `add r8,r1,r9`

36: `xor r10,r1,r11`

We don’t know yet the instruction being executed is a branch. Fetch the branch successor.

Now, target address is available.

Now, we know the instruction being executed is a branch. But stall until branch target address is known.
Summary (1)

- **CISC vs. RISC**
  - CISC: easy for compiler, fewer code bytes
  - RISC: better for optimizing compilers, can make run fast with simple chip design

- **CISC vs. RISC: Current status**
  - For desktop processors, choice of ISA not a technical issue
    - With enough hardware, can make anything run fast
    - Code compatibility more important
  - For embedded processors, RISC makes sense
    - Smaller, cheaper, less power
Summary (2)

- **Pipelining**
  - Improved throughput

- **Problems in pipelining**
  - Structural hazards
  - Data hazards
  - Control hazards

- **Instruction set design affects complexity of pipeline implementation**