6.823 SPring15 
Quiz 1 Review (L01-L05)

Hsin-Jung Yang
EDSAC

• Accumulator based
• Absolute addressing of memory
  – No index registers
• Use self-modifying code for indirect accesses and subroutine calls
**EDSACjr**

- A simplified version of EDSAC instruction set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD ( n )</td>
<td>Accum ← Accum + M[( n )]</td>
</tr>
<tr>
<td>SUB ( n )</td>
<td>Accum ← Accum - M[( n )]</td>
</tr>
<tr>
<td>LD ( n )</td>
<td>Accum ← M[( n )]</td>
</tr>
<tr>
<td>ST ( n )</td>
<td>M[( n )] ← Accum</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Accum ← 0</td>
</tr>
<tr>
<td>OR ( n )</td>
<td>Accum ← Accum</td>
</tr>
<tr>
<td>AND ( n )</td>
<td>Accum ← Accum &amp; M[( n )]</td>
</tr>
<tr>
<td>SHIFTR ( n )</td>
<td>Accum ← Accum siftr ( n )</td>
</tr>
<tr>
<td>SHIFTL ( n )</td>
<td>Accum ← Accum shiftl ( n )</td>
</tr>
<tr>
<td>BGE ( n )</td>
<td>If Accum ≥ 0 then PC ← ( n )</td>
</tr>
<tr>
<td>BLT ( n )</td>
<td>If Accum &lt; 0 then PC ← ( n )</td>
</tr>
<tr>
<td>END</td>
<td>Halt machine</td>
</tr>
</tbody>
</table>
Programming with EDSACjr

\[ C_i \leftarrow A_i + B_i, \ 1 \leq i \leq n \]

```
C
1

ONE

i \leftarrow i + 1, \ 1 \leq i \leq n

A

B

C

N

F1

F2

F3

LD
ADD
ST
LD
ADD
ST
LD
ADD
ST
LD
ADD
ST
CLEAR
BGE
DONE

LD
ADD
ST
N
DONE
ONE
N
A
B
C
F1
ONE
F1
F2
ONE
F2
F3
ONE
F3
LOOP
END
```

DONE

END
Programming with EDSACjr

• Subroutine call in EDSACjr

Just before jumping to the subroutine, the caller loads the return address into the accumulator.

The callee stores the return address.
MIPS 5-Stage Pipeline
Pipeline Hazards

• **Structural Hazard**
  – An instruction in the pipeline may need a resource being used by another instruction in the pipeline

• **Data Hazard**
  – An instruction’s read depends on the data produced by an earlier instruction

• **Control Hazard**
  – Next PC calculation depends on the instructions in the pipeline (branches, exceptions)
Pipeline Hazards

• Structural Hazard
  – An instruction in the pipeline may need a resource being used by another instruction in the pipeline
  – Ex: Princeton-style architecture
  – Solution: Stall
Pipeline Hazards

• Data Hazard
  – An instruction’s read depends on the data produced by an earlier instruction
  – EX: read-after-write (RAW) data hazard
    \[
    r4 \leftarrow r1 + 5 \\
    r3 \leftarrow r4 + 10
    \]
  – Solution: stall or bypass
Resolving Data Hazards By Stalling
Resolving Data Hazards By Stalling

$$C_{\text{dest}}$$

$$ws = \text{Case opcode}$$
- ALU $$\Rightarrow$$ rd
- ALUi, LW $$\Rightarrow$$ rt
- JAL, JALR $$\Rightarrow$$ R31

$$we = \text{Case opcode}$$
- ALU, ALUi, LW $$\Rightarrow$$ (ws $\neq$ 0)
- JAL, JALR $$\Rightarrow$$ on
- ... $$\Rightarrow$$ off

$$C_{\text{re}}$$

$$re1 = \text{Case opcode}$$
- ALU, ALUi,
- LW, SW, BZ,
- JR, JALR $$\Rightarrow$$ on
- J, JAL $$\Rightarrow$$ off

$$re2 = \text{Case opcode}$$
- ALU, SW $$\Rightarrow$$ on
- ... $$\Rightarrow$$ off

$$C_{\text{stall}}$$

$$\text{stall} = ((rs_D = ws_E) \cdot we_E +$$
$$\quad (rs_D = ws_M) \cdot we_M +$$
$$\quad (rs_D = ws_W) \cdot we_W) \cdot re1_D +$$
$$\quad ((rt_D = ws_E) \cdot we_E +$$
$$\quad (rt_D = ws_M) \cdot we_M +$$
$$\quad (rt_D = ws_W) \cdot we_W) \cdot re2_D$$
Resolving Data Hazards By Bypassing

\[ r1 \leftarrow M[r2+5] \]
\[ r3 \leftarrow r1 + 10 \]

\[ \text{stall} = (rs_D = ws_E) \cdot (\text{opcode}_E = \text{LW}_E) \cdot (ws_E \neq 0) \cdot \text{re1}_D \\
+ (rt_D = ws_E) \cdot (\text{opcode}_E = \text{LW}_E) \cdot (ws_E \neq 0) \cdot \text{re2}_D \]
Resolving Data Hazards By Bypassing

Bypass priority: Bypass \( E \rightarrow D \) > Bypass \( M \rightarrow D \) > Bypass \( W \rightarrow D \)
Pipeline Hazards

• **Control Hazard**
  - Next PC calculation depends on the instructions in the pipeline (ex: branches, exceptions)
  - Solution1: stall
  - Solution2: speculate
    - Guessed correctly ➔ do nothing
    - Guessed incorrectly ➔ kill and restart
Quiz 1 Handout (HAL 180)

- **HAL 180**
  - Use condition codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign Flag (SF)</td>
<td>Stores 1 if the result of the last arithmetic or comparison instruction was negative, 0 if it was positive</td>
</tr>
<tr>
<td>Zero Flag (ZF)</td>
<td>Stores 1 if the result of the last arithmetic, logical, or comparison instruction was zero, and 0 if it was non-zero</td>
</tr>
</tbody>
</table>

Table 1. HAL 180 status flags.
### ISA

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>SF</th>
<th>ZF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic Instructions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD $s_1, s_2$</td>
<td>$s_1 \leftarrow s_1 + s_2$</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>SUB $s_1, s_2$</td>
<td>$s_1 \leftarrow s_1 - s_2$</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>MUL $s_1, s_2$</td>
<td>$s_1 \leftarrow s_1 \times s_2$</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td><strong>Logical Instructions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND $s_1, s_2$</td>
<td>$s_1 \leftarrow s_1 &amp; s_2$</td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>OR $s_1, s_2$</td>
<td>$s_1 \leftarrow s_1 \lor s_2$</td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>XOR $s_1, s_2$</td>
<td>$s_1 \leftarrow s_1 \oplus s_2$</td>
<td></td>
<td>W</td>
</tr>
<tr>
<td><strong>Comparison Instructions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP $s_1, s_2$</td>
<td>$\text{temp} \leftarrow s_1 - s_2$</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td><strong>Jump Instructions</strong></td>
<td>jump to the address specified by target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP $\text{target}$</td>
<td>jump to $\text{target}$ if SF == 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JL $\text{target}$</td>
<td>jump to $\text{target}$ if SF == 0 and ZF == 0</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Memory Instructions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD $s_1, s_2$</td>
<td>$s_1 \leftarrow M[s_2]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST $s_1, s_2$</td>
<td>$M[s_1] \leftarrow s_2$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. HAL 180 instruction set.
HAL 180: 6-stage Pipeline