Midterm Review

Jason Mars
ISA - Instruction Length

**Fixed Length**

32 bits

- **opcode**
- **rs**
- **rt**
- **rd**
- **shft**
- **amount**
- **fnc**

32 bits

- **opcode**
- **rs**
- **rt**
- **immediate / offset**

32 bits

- **opcode**
- **target**

32 bits

- **opcode**
- **rs**
- **rt**
- **rd**
- **shft**
- **amount**
- **fnc**

32 bits

- **opcode**
- **target**

32 bits

- **opcode**
- **rs**
- **rt**
- **immediate / offset**

**Variable Length**

a. **JE EIP + displacement**

<table>
<thead>
<tr>
<th>JE</th>
<th>Condition</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

b. **CALL**

<table>
<thead>
<tr>
<th>CALL</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

c. **MOV EBX, [EDI + 45]**

<table>
<thead>
<tr>
<th>MOV</th>
<th>d</th>
<th>w</th>
<th>0m</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

d. **PUSH ESI**

<table>
<thead>
<tr>
<th>PUSH</th>
<th>Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

e. **ADD EAX, #6765**

<table>
<thead>
<tr>
<th>ADD</th>
<th>Reg</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

f. **TEST EDX, #42**

<table>
<thead>
<tr>
<th>TEST</th>
<th>Postbyte</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
Why 5 bits for rs, rt, rd?
What impacts # bits in opcode
Why is it ok to have 16 bits in immediate
What is the difference between immediate and register source
What are the trade-offs between fixed length and variable length?
What is a load-store ISA?
Addressing Modes

- Register direct   R3
- Immediate (literal)  #25
- Direct (absolute) M[10000]

- Register indirect   M[R3]
- Base+Displacement M[R3 + 10000]
- Base+Index   M[R3 + R4]
- Scaled Index M[R3 + R4*d + 10000]
- Autoincrement   M[R3++]
- Autodecrement   M[R3 - -]

- Memory Indirect M[ M[R3] ]
ISA - Addressing Modes in MIPS

**register direct**

```
| OP | rs | rt | rd | sa | funct |
```

add $1, $2, $3

**immediate**

```
| OP | rs | rt | immediate |
```

add $1, $2, #35

base + displacement

lw $1, disp($2)

\[(R1 = M[R2 + disp])\]

We get register indirect and absolute for free, but how?

**register indirect**

\[\Rightarrow \text{disp} = 0\]

**absolute**

\[\Rightarrow (rs) = 0\]
Memory Organization

- What is a word?
- What does it mean to be word aligned?
- What are the values of the last two bits of a word aligned address?
Performance

Speedup \(\frac{X}{Y}\) = \(\frac{\text{Execution Time}_Y}{\text{Execution Time}_X}\) = n
Performance

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Performance

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[
\text{Speedup } \frac{X}{Y} =
\]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Performance

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[
\begin{align*}
X &= \text{machine B} \\
\text{Speedup } \frac{X}{Y} &= \\
Y &= \text{machine A}
\end{align*}
\]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Performance

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[
X = \text{machine B} \\
\text{Speedup } (X/Y) = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{9}{6} = 1.5x
\]

\[
Y = \text{machine A}
\]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Performance

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[
\text{Speedup } \frac{X}{Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{9}{6} = 1.5x
\]

\[
X = \text{machine B}
\]
\[
Y = \text{machine A}
\]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

\[
\text{Speedup } \frac{X}{Y} = \frac{9}{3} = 3x
\]
Performance

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[
\begin{align*}
\text{X} &= \text{machine B} \\
\text{Y} &= \text{machine A} \\
\text{Speedup} \ (X/Y) &= \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n \ 1.5x \\
\text{Execution Time}_Y &= 9 \\
\text{Execution Time}_X &= 6
\end{align*}
\]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

\[
\begin{align*}
\text{X} &= \text{machine B} \\
\text{Y} &= \text{machine A} \\
\text{Speedup} \ (X/Y) &= \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n \ 3x \\
\text{Execution Time}_Y &= 9 \\
\text{Execution Time}_X &= 3
\end{align*}
\]
Performance - Execution Time

- Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

\[
\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]
Performance - Execution Time

- Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

\[
\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

\[
500,000 * 
\]
Performance - Execution Time

• Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

\[
\text{Execution Time} = \left( \frac{\text{Instructions}}{\text{Program}} \right) \times \left( \frac{\text{Cycles}}{\text{Instruction}} \right) \times \left( \frac{\text{Seconds}}{\text{Cycle}} \right)
\]

\[
500,000 \times (0.2 \times 6 + 0.8 \times 1)
\]
Performance - Execution Time

- Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

\[
\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

\[
500,000 \times (.2 \times 6 + .8 \times 1) \times 1 \text{ ns} =
\]
Performance - Execution Time

• Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

\[
\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

\[
500,000 \times (0.2 \times 6 + 0.8 \times 1) \times 1 \text{ ns} = 1,000,000 \text{ ns}
\]
Amdahl’s Law

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

total execution time = 1

\[
\text{Fraction}_{\text{enhanced}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \quad \text{Monday, February 11, 13}
\]
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{1}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{\text{Speedup}}
\]
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.25)}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \text{Speedup}_{\text{enhanced}}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{1}}
\]
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}
\]
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}} = 1.29
\]
ALU - Two’s Complement Representation

- 2’s complement representation of negative numbers
- Take the bitwise inverse and add 1
- Biggest 4-bit Binary Number: 7
  Smallest 4-bit Binary Number: -8

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Two’s Complement Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>1000</td>
</tr>
<tr>
<td>-7</td>
<td>1001</td>
</tr>
<tr>
<td>-6</td>
<td>1010</td>
</tr>
<tr>
<td>-5</td>
<td>1011</td>
</tr>
<tr>
<td>-4</td>
<td>1100</td>
</tr>
<tr>
<td>-3</td>
<td>1101</td>
</tr>
<tr>
<td>-2</td>
<td>1110</td>
</tr>
<tr>
<td>-1</td>
<td>1111</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
</tbody>
</table>
• How many bits does this ALU support?
• What does Binvert do?
• What about CarryIn, CarryOut?
• How do we make an ALU that supports 32 bits?
Thus, if we add hardware to test if the result is 0, we can test for equality. The simplest way is to OR all the outputs together and then send that signal through an inverter:

Figure B.5.12 shows the revised 32-bit ALU. We can think of the combination of the 1-bit Ainvert line, the 1-bit Binvert line, and the 2-bit Operation lines as 4-bit control lines for the ALU, telling it to perform add, subtract, AND, OR, or set on less than. Figure B.5.13 shows the ALU control lines and the corresponding ALU operation.

**FIGURE B.5.12** The final 32-bit ALU.

This adds a Zero detector to Figure B.5.11.
B.5 Constructing a Basic Arithmetic Logic Unit

Thus, if we add hardware to test if the result is 0, we can test for equality. The simplest way is to OR all the outputs together and then send that signal through an inverter:

Figure B.5.12 shows the revised 32-bit ALU. We can think of the combination of the 1-bit Ainvert line, the 1-bit Binvert line, and the 2-bit Operation lines as 4-bit control lines for the ALU, telling it to perform add, subtract, AND, OR, or set on less than. Figure B.5.13 shows the ALU control lines and the corresponding ALU operation.
Thus, if we add hardware to test if the result is 0, we can test for equality. The simplest way is to OR all the outputs together and then send that signal through an inverter:

Figure B.5.12 shows the revised 32-bit ALU. We can think of the combination of the 1-bit Ainvert line, the 1-bit Binvert line, and the 2-bit Operation lines as 4-bit control lines for the ALU, telling it to perform add, subtract, AND, OR, or set on less than. Figure B.5.13 shows the ALU control lines and the corresponding ALU operation.
Full ALU

what signals accomplish:

Binvert  CIn  Oper
add?
sub?
and?
or?
beq?
slt?

FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12.
This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.

module MIPSALU (ALUctl, A, B, ALUOut, Zero);
input [3:0] ALUctl;
input [31:0] A, B;
output reg [31:0] ALUOut;
output Zero;
assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
always @(ALUctl, A, B) begin //reevaluate if these change
case (ALUctl)
  0: ALUOut <= A & B;
  1: ALUOut <= A | B;
  2: ALUOut <= A + B;
  6: ALUOut <= A - B;
  7: ALUOut <= A < B ? 1 : 0;
  12: ALUOut <= ~(A | B); // result is nor
  default: ALUOut <= 0;
endcase
end
endmodule

FIGURE C.5.15 A Verilog behavioral definition of a MIPS ALU.
Full ALU

what signals accomplish:

<table>
<thead>
<tr>
<th>Binvert</th>
<th>CIn</th>
<th>Oper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

add? sub? and? or? beq? slt?
Full ALU

what signals accomplish:
add? 0 0 10
sub? 1 1 10
and? or? beq?
slt?
what signals accomplish:
<table>
<thead>
<tr>
<th>Binvert</th>
<th>CIn</th>
<th>Oper</th>
</tr>
</thead>
<tbody>
<tr>
<td>add?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sub?</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>and?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>or?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>slt?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Full ALU

what signals accomplish:

<table>
<thead>
<tr>
<th></th>
<th>Binvert</th>
<th>CIn</th>
<th>Oper</th>
</tr>
</thead>
<tbody>
<tr>
<td>add?</td>
<td>0 0 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub?</td>
<td>1 1 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and?</td>
<td>0 0 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or?</td>
<td>0 0 01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slt?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Full ALU

what signals accomplish:

<table>
<thead>
<tr>
<th>Binvert</th>
<th>CIn</th>
<th>Oper</th>
</tr>
</thead>
<tbody>
<tr>
<td>add?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sub?</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>and?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>or?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq?</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>slt?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12. This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.

module MIPSALU (ALUctl, A, B, ALUOut, Zero);
input [3:0] ALUctl;
input [31:0] A,B;
output reg [31:0] ALUOut;
output Zero;
assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
always @(ALUctl, A, B) begin //reevaluate if these change
  case (ALUctl)
    0: ALUOut <= A & B;
    1: ALUOut <= A | B;
    2: ALUOut <= A + B;
    6: ALUOut <= A - B;
    7: ALUOut <= A < B ? 1 : 0;
    12: ALUOut <= ~(A | B); // result is nor
    default: ALUOut <= 0;
  endcase
end
endmodule

FIGURE C.5.15 A Verilog behavioral definition of a MIPS ALU.
Full ALU

what signals accomplish:

<table>
<thead>
<tr>
<th></th>
<th>Binvert</th>
<th>CIn</th>
<th>Oper</th>
</tr>
</thead>
<tbody>
<tr>
<td>add?</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>sub?</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>and?</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>or?</td>
<td>0</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>beq?</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>slt?</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

FIGURE C.5.15 A Verilog behavioral definition of a MIPS ALU.
The R-Format (e.g. add) Datapath

add r1, r2, r3
The Load Datapath

lw r1, 1000(r2)
The Store Datapath

\[ \text{sw r1, 1000(r2)} \]
The Branch (beq) Datapath

beq r1, r2, 1000
Single Cycle CPU

- What are the signals?
- What are the signals for...
  - add r1, r2, r3
  - lw r1, 1000(r3)
  - sw r2, 45(r1)
  - beq r1, r2, Loop
Complete Multi-Cycle Datapath
1. Instruction Fetch

IR = Memory[PC]
PC = PC + 4
2. Instruction Decode and Register Fetch

A = Register[IR[25-21]]
B = Register[IR[20-16]]
ALUOut = PC + (sign-extend (IR[15-0]) << 2)
if (A == B)  PC = ALUOut
3. Execution (R-Type)

ALUout = A op B
4. R-Type Completion

\[
\text{Reg}[\text{IR}[15-11]] = \text{ALUout}
\]
4. Memory Address Computation

\[
ALU_{out} = A + \text{sign-extend}(IR[15-0])
\]
4. Memory Access Load

memory-data = Memory[ALUout]
4. Memory Access Store

Memory[ALUout] = B
5. Load Write-Back

\[ \text{Reg}[\text{IR}[20-16]] = \text{memory-data} \]
Some Juicy Questions

• How many cycles will it take to execute this code?

```assembly
lw  $t2, 0($t3)
lw  $t3, 4($t3)  
beq $t2, $t3, Label  #assume not taken
add $t5, $t2, $t3
sw  $t5, 8($t3)  
Label: ...
```

• What’s going on during the 8th cycle of execution?

• In what cycle does the actual addition of $t2 and $t3 take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?

```
5 lw $t2, 0($t3)
lw $t3, 4($t3)
beq $t2, $t3, Label #assume not taken
add $t5, $t2, $t3
sw $t5, 8($t3)
Label: ...
```

• What's going on during the 8th cycle of execution?

• In what cycle does the actual addition of $t2 and $t3 take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
beq $t2, $t3, Label  #assume not taken
add $t5, $t2, $t3
sw $t5, 8($t3)
Label: ...

• What's going on during the 8th cycle of execution?

• In what cycle does the actual addition of $t2 and $t3 take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label  #assume not taken
add $t5, $t2, $t3
sw $t5, 8($t3)
Label: ...

• What's going on during the 8th cycle of execution?

• In what cycle does the actual addition of $t2 and $t3 take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?

```assembly
5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label  #assume not taken
4 add $t5, $t2, $t3
sw $t5, 8($t3)
Label: ...
```

• What’s going on during the 8th cycle of execution?

• In what cycle does the actual addition of $t2 and $t3 take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?

```assembly
5  lw $t2, 0($t3)
5  lw $t3, 4($t3)
3  beq $t2, $t3, Label  #assume not taken
4  add $t5, $t2, $t3
4  sw $t5, 8($t3)
Label: ...
```

• What's going on during the 8th cycle of execution?

• In what cycle does the actual addition of $t2 and $t3 take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?

\[
\begin{align*}
5 & \quad \text{lw} \quad t_2, \quad 0(t_3) \\
5 & \quad \text{lw} \quad t_3, \quad 4(t_3) \\
3 & \quad \text{beq} \quad t_2, \quad t_3, \quad \text{Label} \quad \#\text{assume not taken} \\
4 & \quad \text{add} \quad t_5, \quad t_2, \quad t_3 \\
4 & \quad \text{sw} \quad t_5, \quad 8(t_3) \\
\end{align*}
\]

Label: ...

\[21\]

• What’s going on during the 8th cycle of execution?

• In what cycle does the actual addition of \( t_2 \) and \( t_3 \) take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

- How many cycles will it take to execute this code?

```plaintext
5 lw $t2, 0($t3)  
5 lw $t3, 4($t3)  
3 beq $t2, $t3, Label  #assume not taken  
4 add $t5, $t2, $t3  
4 sw $t5, 8($t3)  
Label: ...
```

- What's going on during the 8th cycle of execution?

```plaintext
lw
```

- In what cycle does the actual addition of $t2 and $t3 take place?

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?
  
  5 lw $t2, 0($t3)
  5 lw $t3, 4($t3)
  3 beq $t2, $t3, Label  #assume not taken
  4 add $t5, $t2, $t3
  4 sw $t5, 8($t3)
  Label: ...

• What's going on during the 8th cycle of execution?
  lw

• In what cycle does the actual addition of $t2 and $t3 take place?
  16th

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
Some Juicy Questions

• How many cycles will it take to execute this code?
  
  5 \text{lw} \ $t2, \ 0($t3) \\
  5 \text{lw} \ $t3, \ 4($t3) \\
  3 \text{beq} \ $t2, \ $t3, \ \text{Label} \ #\text{assume not taken} \\
  4 \text{add} \ $t5, \ $t2, \ $t3 \\
  4 \text{sw} \ $t5, \ 8($t3) \\
  \text{Label:} \ ... \\

• What's going on during the 8th cycle of execution?
  
  16th

• In what cycle does the actual addition of $t2 and $t3 take place?
  
  16th

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
  
  .2^* (5) +
Some Juicy Questions

• How many cycles will it take to execute this code?

```assembly
5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label  #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
```

21 cycles

• What's going on during the 8th cycle of execution?

```assembly
lw
```

• In what cycle does the actual addition of $t2 and $t3 take place?

16th cycle

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

\[ .2 \times 5 + .1 \times 4 + \]
Some Juicy Questions

• How many cycles will it take to execute this code?
  5 lw $t2, 0($t3)
  5 lw $t3, 4($t3)
  3 beq $t2, $t3, Label  #assume not taken
  4 add $t5, $t2, $t3
  4 sw $t5, 8($t3)
  Label: ...

  \[ \text{CPI} = 21 \]

• What's going on during the 8th cycle of execution?
  \[ \text{lwan} \]

• In what cycle does the actual addition of $t2 and $t3 take place?
  16th

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
  \[ 0.2 \times 5 + 0.1 \times 4 + 0.5 \times 4 + \]
Some Juicy Questions

• How many cycles will it take to execute this code?
  5  lw  $t2, 0($t3)  
  5  lw  $t3, 4($t3)  
  3  beq  $t2, $t3, Label  #assume not taken  
  4  add  $t5, $t2, $t3  
  4  sw  $t5, 8($t3)  
  Label: ...

  21

• What’s going on during the 8th cycle of execution?

  lw

• In what cycle does the actual addition of $t2 and $t3 take place?

  16th

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

  \[0.2(5) + 0.1(4) + 0.5(4) + 0.2(3) = \]
Some Juicy Questions

• How many cycles will it take to execute this code?

\[ \text{l}w \ \ \text{$t2, 0($t3)$} \]
\[ \text{l}w \ \ \text{$t3, 4($t3)$} \]
\[ \text{beq $t2, $t3, Label} \ ] \text{#assume not taken} \]
\[ \text{add $t5, $t2, $t3} \]
\[ \text{sw $t5, 8($t3)$} \]
Label: ...

21 cycles

• What's going on during the 8th cycle of execution?

lw

• In what cycle does the actual addition of $t2 and $t3 take place?

16th

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

\[ .2*(5) + .1*(4) + .5*(4) + .2*(3) = 4 \]
Good Luck!