A Second Datapath Example
YH16

Lecture 09
Prof. Yih Huang

A 16-Bit Architecture: YH16

- A “word” is 16 bit wide
- 32 general purpose registers, 16 bits each
- Like MIPS, R0 is hardwired zero.
- 16 bit PC
- 16 bit ALU
- Memory space: $2^{16}$ words.
# Instruction Format I

<table>
<thead>
<tr>
<th>opcode</th>
<th>name</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>add</td>
<td>rs ← rs + rt</td>
</tr>
<tr>
<td>000001</td>
<td>and</td>
<td>rs ← rs AND rt</td>
</tr>
<tr>
<td>000010</td>
<td>or</td>
<td>rs ← rs OR rt</td>
</tr>
<tr>
<td>000011</td>
<td>xor</td>
<td>rs ← rs XOR rt</td>
</tr>
<tr>
<td>001000</td>
<td>sub</td>
<td>rs ← rs - rt</td>
</tr>
<tr>
<td>001100</td>
<td>slt</td>
<td>rs ← 1 if rs&lt;rt, or 0 otherwise</td>
</tr>
</tbody>
</table>

# Instruction Format II

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opcode</th>
<th>name</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>addi</td>
<td>rt ← rs + Immd16</td>
</tr>
<tr>
<td>010001</td>
<td>andi</td>
<td>rt ← rs AND Immd16</td>
</tr>
<tr>
<td>010010</td>
<td>ori</td>
<td>rt ← rs OR Immd16</td>
</tr>
<tr>
<td>010011</td>
<td>xori</td>
<td>rt ← rs XOR Immd16</td>
</tr>
<tr>
<td>011000</td>
<td>subi</td>
<td>rt ← rs - Immd16</td>
</tr>
</tbody>
</table>
Memory Reference Instructions

<table>
<thead>
<tr>
<th>opcode</th>
<th>name</th>
<th>format</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>ld</td>
<td>I</td>
<td>rt ← Mem[rs]</td>
</tr>
<tr>
<td>100001</td>
<td>ldi</td>
<td>II</td>
<td>rt ← Mem[rs+Immd16]</td>
</tr>
<tr>
<td>100010</td>
<td>st</td>
<td>I</td>
<td>Mem[rs] ← rt</td>
</tr>
<tr>
<td>100011</td>
<td>sti</td>
<td>II</td>
<td>Mem[rs+Immd16] ← rt</td>
</tr>
</tbody>
</table>

Branch Instructions

<table>
<thead>
<tr>
<th>opcode</th>
<th>name</th>
<th>format</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>110000</td>
<td>bt</td>
<td>I</td>
<td>PC ← rs, if rt ≠ 0</td>
</tr>
<tr>
<td>110001</td>
<td>bf</td>
<td>I</td>
<td>PC ← rs, if rt = 0</td>
</tr>
<tr>
<td>110100</td>
<td>beq</td>
<td>II</td>
<td>PC ← PC+Immd16, if rs = rt</td>
</tr>
<tr>
<td>110101</td>
<td>bne</td>
<td>II</td>
<td>PC ← PC+Immd16, if rs ≠ rt</td>
</tr>
</tbody>
</table>

- bt: branch when true
- bf: branch when false
- Unconditional branches?
Exercise

Create a program that performs

\[ \text{Mem[F0F2]} = \text{Mem[F0F0]} + \text{Mem[F0F1]} \]

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td></td>
</tr>
<tr>
<td>1008</td>
<td></td>
</tr>
<tr>
<td>100A</td>
<td></td>
</tr>
</tbody>
</table>

Exercise

Create a program that jumps to location ABCD when Mem[2000] is zero.

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td></td>
</tr>
<tr>
<td>1008</td>
<td></td>
</tr>
<tr>
<td>100A</td>
<td></td>
</tr>
</tbody>
</table>
Processor Memory Interface

Data Path
Can’t write to PC if PCw==0.

When PCw==1

★ write PC if UseZ==00
★ write PC if UseZ==01 and Z==1
★ write PC if UseZ==10 and Z==0

Input to PC determined by PCx

Control Signals

ALUop (4 bits)

<table>
<thead>
<tr>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>1011</th>
<th>1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>OR</td>
<td>XOR</td>
<td>ADD</td>
<td>SUB</td>
<td>SLT</td>
</tr>
</tbody>
</table>

opcode ≡ IR[15:10]
rs ≡ IR[9:5]
rt ≡ IR[4:0]
Multiplexer controls:
- ✪1x (mux for the 1st input of ALU),
- ✪2x (mux for the 2nd input of ALU),
- ✪RFx (mux for register file write),
- ✪PCx (mux for PC)
- ✪MAx (mux for memory address)
- ✪MDx (mux for memory data)

Write enable controls: Aw, Bw, Cw, PCw, RFw, IRw, MDRw, RW

- UseZ: whether to use the Zero output from ALU to affect PC writes.
- UseR0: whether the first read register # is rs or 0.
- UseRt: whether the write register # is rs or rt.
Discussion

- Notice the lack of Zw (enable-write-to-Z)
- Z is written in all cycles.
- This creates constraints in timing.
- Example, to see A==B
  - There is only one cycle where Z reflects the result of A−B.
  - That cycle follows the cycle of A-B.
- You must take into account these constraints when designing cycle by cycle activities of instructions.

Exercise

- Give the control signals to fetch the next instruction and increase PC at the same time.
  - IR ← Mem[PC]
  - C ← PC + 1

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise

- Give the control signals to carry out the following tasks in one cycle:
  - \( C \leftarrow PC - C \)
  - \( A \leftarrow R0 \)
  - \( Rt \leftarrow PC \)

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td>P</td>
<td>C</td>
<td>R</td>
<td>F</td>
</tr>
<tr>
<td>I</td>
<td>R</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>M</td>
<td>D</td>
<td>R</td>
<td>t</td>
</tr>
</tbody>
</table>

ADD

- Cycle 0:
  - \( IR \leftarrow \text{Mem}[PC] \)
  - \( C \leftarrow PC + 1 \)

- Cycle 1:
  - \( PC \leftarrow C \)
  - \( A \leftarrow \text{Reg}[rs] \)
  - \( B \leftarrow \text{Reg}[rt] \)

- Cycle 2:
  - \( C \leftarrow A + B \)

- Cycle 3:
  - \( \text{Reg}[rs] \leftarrow C \)

Instructions and, or, xor, sub, slt are similar.
Hints

- This is not a programming exercise.
- Activities in each cycle must be doable with the datapath without conflicts in resources.
- Parallel activities are not mandatory, that is, one-cycle-one-action is not wrong.
- In practice, parallel activities are highly desired in order to reduce the numbers of cycles per instruction.

Exercise

- Give the control signals to execute ADD.

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>R</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A  B  C  P  R  I  M</td>
<td>1</td>
<td>2</td>
<td>RF</td>
</tr>
</tbody>
</table>
ADDI

- Cycle 0:
  - \( \star IR \leftarrow \text{Mem}[PC] \)
  - \( \star C \leftarrow PC + 1 \)

- Cycle 1:
  - \( \star PC \leftarrow C \)
  - \( \star A \leftarrow \text{Reg}[rs] \)
  - \( \star MDR \leftarrow \text{Mem}[C] \)

- Cycle 2:
  - \( \star C \leftarrow A + MDR \)

- Cycle 3:
  - \( \star \text{Reg}[rt] \leftarrow C \)
  - \( \star C \leftarrow PC + 1 \)

- Cycle 4:
  - \( \star PC \leftarrow C \)

andi, ori, xori, subi are similar

Exercise

- Give the control signals to execute ADDI.

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>RF</td>
</tr>
</tbody>
</table>
LD (Load)

- **Cycle 0:**
  - $\star$IR $\leftarrow$ Mem[PC]
  - $\star$C $\leftarrow$ PC + 1

- **Cycle 1:**
  - $\star$A $\leftarrow$ Reg[rs]
  - $\star$PC $\leftarrow$ C

- **Cycle 2:**
  - $\star$MDR $\leftarrow$ Mem[A]

- **Cycle 3:**
  - $\star$Reg[rt] $\leftarrow$ MDR

Exercise: ST (Store)
## LDI (Load with Immd)

- **Cycle 0:**
  - \( IR \leftarrow \text{Mem}[PC] \)
  - \( C \leftarrow PC + 1 \)
- **Cycle 1:**
  - \( A \leftarrow \text{Reg}[rs] \)
  - \( MDR \leftarrow \text{Mem}[C] \)
  - \( PC \leftarrow C \)
- **Cycle 2:**
  - \( C \leftarrow A + MDR \)
- **Cycle 3:**
  - \( C \leftarrow PC + 1 \)
  - \( MDR \leftarrow \text{Mem}[C] \)
- **Cycle 4:**
  - \( \text{Reg}[rt] \leftarrow MDR \)
  - \( PC \leftarrow C \)

## Exercise

- **Give the cycle by cycle actions of STI**
Exercise

- Give the control signals to execute STI.

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A  B  C  P  F I  R  M  D  R</td>
<td>R  W  1  2  RF  P  C  MA  M  D</td>
<td>Z  R  0  R  t</td>
</tr>
</tbody>
</table>

BT (branch when true)

- Cycle 0:
  - $\star$IR $\leftarrow$ Mem[PC]
  - $\star$C $\leftarrow$ PC + 1

- Cycle 1:
  - $\star$PC $\leftarrow$ C
  - $\star$A $\leftarrow$ Reg[0]
  - $\star$B $\leftarrow$ Reg[rt]

- Cycle 2:
  - $\star$Z $\leftarrow$ (A==B)
  - $\star$A $\leftarrow$ Reg[rs]

- Cycle 3:
  - $\star$PC $\leftarrow$ A, if $\overline{Z}$
**BEQ**

- **Cycle 0:**
  - $\star IR \leftarrow Memory[PC]$
  - $\star C \leftarrow PC + 1$

- **Cycle 1:**
  - $\star PC \leftarrow C$
  - $\star MDR \leftarrow Mem[C]$
  - $\star A \leftarrow Reg[rs]$
  - $\star B \leftarrow Reg[rt]$

- **Cycle 2:** $C \leftarrow PC + 1$
- **Cycle 3:** $PC \leftarrow C$
- **Cycle 4:**
  - $\star C \leftarrow PC + MDR$

- **Cycle 5:**
  - $\star Z \leftarrow (A==B)$

- **Cycle 6:**
  - $\star PC \leftarrow C$, if $Z$

---

**Exercises**

- **Give the control signals of BEQ cycle 6**

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>RW</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C P C R F I R M D R</td>
<td>1 2 RF P C MA M D Z R 0 R t</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Give the control signals of BT cycle 3.**

<table>
<thead>
<tr>
<th>ALU</th>
<th>Write Enables</th>
<th>RW</th>
<th>Multiplexer Controls</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C P C R F I R M D R</td>
<td>1 2 RF P C MA M D Z R 0 R t</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Control Unit

- The control unit is responsible for generating control signals so that the datapath carries out right actions at right times.
- We use a 3-bit register S to keep track of the present cycle in executing the current instruction (2\textsuperscript{nd} cycle of add, 5\textsuperscript{th} of ldi, …)
  \( S \) is just three D flip-flops.

Generating Control Signals

- The conditions when PCw is true:
  - \( \star \) Cycle 1 of all instructions
  - \( \star \) Cycle 3 of bt, bf
  - \( \star \) Cycle 4 of addi, andi, ori, xori, subi
  - \( \star \) Cycle 4 of ldi, sti
  - \( \star \) Cycle 6 of beq, bne
Boolean expression to generate PCw:

\[
S_2S_1S_0 + \\
S_2S_1S_0 \left( \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} + \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} \right) + \\
S_2S_1S_0 \left( \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} + \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} \right) + \\
S_2S_1S_0 \left( \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} + \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} \right) + \\
S_2S_1S_0 \left( \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} + \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} \right) + \\
S_2S_1S_0 \left( \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} + \text{I}_{15}\text{I}_{14}\text{I}_{13}\text{I}_{12}\text{I}_{11}\text{I}_{10} \right)
\]

Exercise

Give the conditions when ALU\textsubscript{3} is true:

Boolean expression to generate ALU\textsubscript{3}:
Exercise

- Give the conditions when UseRt is true:

- Boolean expression to generate UseRt:

Exercise

- Give the conditions when MAx0 is true:

- Give the conditions when MAx1 is true:
Output Function

- Figure out the Boolean expressions for all 25 control signals, simplify them, and draw digital circuit accordingly.
- The result is the output function.

- Next we work on the next state function, for which we need to figure out the state transition diagram first.
The conditions when $NS_0$ is true

The Boolean expression for $NS_0$
Hardwired Implementations

- Write down Boolean expressions for control signals and next states
- Simplify the expressions and draw digital circuits.

- Or one can use a more systematic approach.

A Truth Table for YH16 Controls

<table>
<thead>
<tr>
<th>6-bit for opcode</th>
<th>3-bit state</th>
<th>25-bit control signals</th>
<th>3-bit next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>oooooo</td>
<td>sss</td>
<td>ccccccc</td>
<td>sss</td>
</tr>
</tbody>
</table>
ROM Implementations

- Just burn the truth table to ROM.
- Address = opcode and state (cycle #)
- If we limit all instructions to 8 steps, then YH16 needs a ROM with
  - 9 bit address (3-bit next state; 6 bits opcode)
  - Each ROM word has $25 + 3 = 28$ bits

ROM Addresses and Contents

- 9 bit Address
- 25-bit for control signals
- 3-bit for next state
- 3-bit for state
- 6-bit for opcode
- 28 bit contents
Microprogramming

- It is as if every instruction (opcode) has a dedicated “subroutine,” limited to 8 steps.
- They are of course not real subroutines (at least not the kinds we learn in 112 to 310).
- Always keep in mind that every step corresponds to one cycle time’s worth of hardware actions.
- What kinds of actions allowed (not allowed) entirely depend on the particular datapath.
Instruction Execution Flow

- Consider the executions of
  - Or r2, r3
  - Add r1, r2

Can you see the flow of microinstructions?

Microcode: Trade-offs

- Pros
  - Easy to design and write
  - Design architecture and microcode in parallel
  - Easy to make changes after chips are sold (to fix bugs for example)

- Cons
  - Slow
  - Use more silicon areas than simplified Boolean expressions.
How is a PC powered up?

- The processor receives a RESET signal
- It also sent to the processor when you push the reset button.
- RESET is one of the inputs to CU
- Upon reset a processor executes the instruction at a fixed memory location.
  - The exact location depends on the architecture.

Bootstrapping in IA32

- A pin is dedicated to an external signal called RESET.
- When receiving RESET, an x86 processor is hardwired to jump to address 0xFFFFFFF0.
- The address has to be part of ROM.
- The set of programs in ROM is called Basic Input and Output Systems, or BIOS.
Once We Enter BIOS

- Perform a set of hardware test called Power-On Self Test, or POST.
- Initialize hardware and display installed devices.
- Look for booting devices.
- Load the OS from a booting device.

Booting/Resetting YH16

- Upon receiving a reset signal
  - *Abort the current instruction
  - *Set PC to jump to location 0000.
- Precisely:
  - *PC ← 0000_{16}
  - *Next state ← 000_{2}
  - *IR ?
- How is this achieved in the control unit?
D Flip-Flop with Reset and Write-Enable