Announcements

- Homework #3 due Thurs.

- Homework #4 out this Thurs.
  - Due next Thurs.
Key 61c Concept: “Stored Program” Computer

- Instructions and data stored in memory.
- Only difference between two applications (for example, a text editor and a video game), is the sequence of instructions.
- To run a new program:
  - No rewiring required
  - Simply store new program in memory
  - The processor hardware executes the program:
    - fetches (reads) the instructions from memory in sequence
    - performs the specified operation
  - The program counter (PC) keeps track of the current instruction.

Interpreting Machine Code

- Start with opcode
- Opcode tells how to parse the remaining bits
- If opcode is all 0’s
  - R-type instruction
  - Function bits tell what instruction it is
- Otherwise
  - Opcode tells what instruction it is

A processor is a machine code interpreter build in hardware!
Processor Microarchitecture Introduction

Microarchitecture: how to implement an architecture in hardware

Good examples of how to put principles of digital design to practice.

Introduction to final project.

MIPS Processor Architecture

- For now we consider a subset of MIPS instructions:
  - R-type instructions: `and, or, add, sub, slt`
  - Memory instructions: `lw, sw`
  - Branch instructions: `beq`
- Later we’ll add `addi` and `j`
MIPS Microarchitecture Organization
Datapath + Controller + External Memory

How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) ® datapath requirements
   - meaning of each instruction is given by the data transfers (register transfers)
   - datapath must include storage element for ISA registers
   - datapath must support every possible data transfer

2. Select set of datapath components and establish clocking methodology

3. Assemble datapath meeting requirements

4. Analyze implementation of each instruction to determine setting of control points that affect the data transfer.

5. Assemble the control logic.
Review: The MIPS Instruction Formats

R-type

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
<td></td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
<td></td>
</tr>
</tbody>
</table>

I-type

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>address/immediate</td>
<td></td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
<td></td>
</tr>
</tbody>
</table>

J-type

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>target address</td>
<td></td>
</tr>
<tr>
<td>6 bits</td>
<td>26 bits</td>
<td></td>
</tr>
</tbody>
</table>

The different fields are:
op: operation ("opcode") of the instruction
rs, rt, rd: the source and destination register specifiers
shamt: shift amount
funct: selects the variant of the operation in the "op" field
address / immediate: address offset or immediate value
target address: target address of jump instruction

Subset for Lecture

add, sub, or, slt
• addu rd,rs,rt
• subu rd,rs,rt

lw, sw
• lw rt,rs,imm16
• sw rt,rs,imm16

beq
• beq rs,rt,imm16
Register Transfer Descriptions

All start with instruction fetch, e.g.:
\{op , rs , rt , rd , shamt , funct\} ← IMEM[ PC ]
\{op , rs , rt , Imm16\} ← IMEM[ PC ]

```
inst   Register Transfers
add    R[rd] ← R[rs] + R[rt];   PC ← PC + 4
sub    R[rd] ← R[rs] − R[rt];   PC ← PC + 4
or     R[rd] ← R[rs] | R[rt];   PC ← PC + 4
slt    R[rd] ← (R[rs] < R[rt]) ? 1 : 0;   PC ← PC + 4
lw     R[rt] ← DMEM[ R[rs] + sign_ext(Imm16) ];   PC ← PC + 4
sw     DMEM[ R[rs] + sign_ext(Imm16) ] ← R[rt];   PC ← PC + 4
beq    if ( R[rs] == R[rt] ) then PC ← PC + 4 + \{sign_ext(Imm16), 00\}
        else PC ← PC + 4
```

Microarchitecture

Multiple implementations for a single architecture:

- Single-cycle  
  - Each instruction executes in a single clock cycle.

- Multicycle  
  - Each instruction is broken up into a series of shorter steps with one step per clock cycle.

- Pipelined (variant on “multicycle”)  
  - Each instruction is broken up into a series of steps with one step per clock cycle
  - Multiple instructions execute at once.
CPU clocking (1/2)

- **Single Cycle CPU:** All stages of an instruction are completed within one long clock cycle.
  - The clock cycle is made sufficient long to allow each instruction to complete all stages without interruption and within one cycle.

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Reg. Write

CPU clocking (2/2)

- **Multiple-cycle CPU:** Only one stage of instruction per clock cycle.
  - The clock is made as long as the slowest stage.

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Reg. Write

- Several significant advantages over single cycle execution:
  - Unused stages in a particular instruction can be skipped, OR
  - Instructions can be pipelined
MIPS State Elements

- Determine everything about the execution status of a processor:
  - **PC** register
  - 32 registers
  - Memory

Note: for these state elements, clock is used for write but not for read (asynchronous read, synchronous write).

Single-Cycle Datapath: lw fetch

- First consider executing lw

\[ R[rt] \leftarrow \text{DMEM}[R[rs] + \text{sign}_{\text{ext}}(\text{Imm}16)] \]

- **STEP 1**: Fetch instruction
Single-Cycle Datapath: \texttt{lw} register read

\[ R[rt] \leftarrow \text{DMEM}[ R[rs] + \text{sign_ext}(\text{Imm16})] \]

- **STEP 2**: Read source operands from register file

Single-Cycle Datapath: \texttt{lw} immediate

\[ R[rt] \leftarrow \text{DMEM}[ R[rs] + \text{sign_ext}(\text{Imm16})] \]

- **STEP 3**: Sign-extend the immediate
Single-Cycle Datapath: lw address

\[ R[rt] \leftarrow DMEM[ R[rs] + sign\_ext(Imm16)] \]

- **STEP 4**: Compute the memory address

Single-Cycle Datapath: lw memory read

\[ R[rt] \leftarrow DMEM[ R[rs] + sign\_ext(Imm16)] \]

- **STEP 5**: Read data from memory and write it back to register file
Single-Cycle Datapath: lw PC increment

- **STEP 6:** Determine the address of the next instruction

\[ \text{PC} \leftarrow \text{PC} + 4 \]

Single-Cycle Datapath: sw

- Write data in rt to memory

\[ \text{DMEM}[R[rs] + \text{sign_ext(Imm16) \} \leftarrow R[rt]] \]
Single-Cycle Datapath: R-type instructions

- Read from rs and rt
- Write ALUResult to register file
- Write to rd (instead of rt)

\[ R[rd] \leftarrow R[rs] \text{ op } R[rt] \]

Single-Cycle Datapath: \textit{beq}

if ( R[rs] == R[rt] ) then \( PC \leftarrow PC + 4 + \{\text{sign_ext(Imm16), 00}\} \)

- Determine whether values in rs and rt are equal
- Calculate branch target address:
  \[ \text{BTA} = (\text{sign-extended immediate} \ll 2) + (\text{PC} + 4) \]
Review: ALU

<table>
<thead>
<tr>
<th>$F_{2:0}$</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>A &amp; B</td>
</tr>
<tr>
<td>001</td>
<td>A</td>
</tr>
<tr>
<td>010</td>
<td>A + B</td>
</tr>
<tr>
<td>011</td>
<td>not used</td>
</tr>
<tr>
<td>100</td>
<td>A &amp; ~B</td>
</tr>
<tr>
<td>101</td>
<td>A</td>
</tr>
<tr>
<td>110</td>
<td>A - B</td>
</tr>
<tr>
<td>111</td>
<td>SLT</td>
</tr>
</tbody>
</table>
Control Unit

Control Unit: ALU Decoder

<table>
<thead>
<tr>
<th>ALUOp1:0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Add</td>
</tr>
<tr>
<td>01</td>
<td>Subtract</td>
</tr>
<tr>
<td>10</td>
<td>Look at Funct</td>
</tr>
<tr>
<td>11</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALUOp1:0</th>
<th>Funct</th>
<th>ALUControl2:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>XXXXX</td>
<td>010 (Add)</td>
</tr>
<tr>
<td>X1</td>
<td>XXXXX</td>
<td>110 (Subtract)</td>
</tr>
<tr>
<td>1X</td>
<td>100000 (add)</td>
<td>010 (Add)</td>
</tr>
<tr>
<td>1X</td>
<td>100010 (sub)</td>
<td>110 (Subtract)</td>
</tr>
<tr>
<td>1X</td>
<td>100100 (and)</td>
<td>000 (And)</td>
</tr>
<tr>
<td>1X</td>
<td>100101 (or)</td>
<td>001 (Or)</td>
</tr>
<tr>
<td>1X</td>
<td>101010 (slt)</td>
<td>111 (SLT)</td>
</tr>
</tbody>
</table>
Control Unit: Main Decoder

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op&lt;sub&gt;5:0&lt;/sub&gt;</th>
<th>RegWrite</th>
<th>RegDst</th>
<th>ALuSrc</th>
<th>Branch</th>
<th>MemWrite</th>
<th>MemtoReg</th>
<th>ALUOP&lt;sub&gt;1:0&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>000000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>lw</td>
<td>100011</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>sw</td>
<td>101011</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>00</td>
</tr>
<tr>
<td>beq</td>
<td>000100</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>01</td>
</tr>
</tbody>
</table>

Single-Cycle Datapath Example: or
Extended Functionality: **addi**

- No need to change to datapath...

**Control Unit: addi**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op[5:0]</th>
<th>RegWrite</th>
<th>RegDst</th>
<th>AluSrc</th>
<th>Branch</th>
<th>MemWrite</th>
<th>MemtoReg</th>
<th>ALUOP[1:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addi</td>
<td>00100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of control unit](image)
Extended Functionality: j

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op&lt;sub&gt;inst&lt;/sub&gt;</th>
<th>RegWrite</th>
<th>RegDst</th>
<th>ALuSrc</th>
<th>Branch</th>
<th>MemWrite</th>
<th>MemtoReg</th>
<th>ALUOp&lt;sub&gt;1:0&lt;/sub&gt;</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>000100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Review: Processor Performance

Program Execution Time

\[
\text{Program Execution Time} = (\# \text{ instructions}) \cdot (\text{cycles/instruction}) \cdot (\text{seconds/cycle})
\]

\[
= (\# \text{ instructions}) \times (\text{CPI}) \times (T_C)
\]
Single-Cycle Performance (1)

• $T_c$ is limited by the critical path (1w)

Single-Cycle Performance (2)

• Single-cycle critical path:

$$T_c = t_{pq_{PC}} + t_{mem} + \max(t_{RF\text{read}}, t_{\text{sext}} + t_{\text{mux}}) + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$
Single-Cycle Performance (3)

- In most implementations, limiting paths are:
  - memory, ALU, register file
  - \( T_c = t_{pq_{PC}} + 2t_{mem} + t_{RF\text{read}} + t_{mux} + t_{ALU} + t_{RF\text{setup}} \)

<table>
<thead>
<tr>
<th>Element</th>
<th>Parameter</th>
<th>Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register clock-to-Q</td>
<td>( t_{pq_{PC}} )</td>
<td>30</td>
</tr>
<tr>
<td>Register setup</td>
<td>( t_{setup} )</td>
<td>20</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>( t_{mux} )</td>
<td>25</td>
</tr>
<tr>
<td>ALU</td>
<td>( t_{ALU} )</td>
<td>200</td>
</tr>
<tr>
<td>Memory read</td>
<td>( t_{mem} )</td>
<td>250</td>
</tr>
<tr>
<td>Register file read</td>
<td>( t_{RF\text{read}} )</td>
<td>150</td>
</tr>
<tr>
<td>Register file setup</td>
<td>( t_{RF\text{setup}} )</td>
<td>20</td>
</tr>
</tbody>
</table>

\[ T_c = t_{pq_{PC}} + 2t_{mem} + t_{RF\text{read}} + t_{mux} + t_{ALU} + t_{RF\text{setup}} \]
Single-Cycle Performance Example

- For a program with 100 billion instructions executing on a single-cycle MIPS processor,

Execution Time =

Pipelined MIPS Processor

- Temporal parallelism
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add pipeline registers between stages
Single-Cycle vs. Pipelined Performance

---

**Single-Cycle**

- Read/Write
- Decode
- Execute
- Write

---

**Pipelined**

- Read/Write
- Decode
- Execute
- Write

---

Pipelining Abstraction

- **Load**: Load immediate or register
- **Add**: Add immediate or register
- **Sub**: Subtract immediate or register
- **Mul**: Multiply immediate or register
- **Div**: Divide immediate or register
- **And**: And immediate or register
- **Or**: Or immediate or register
- **Xor**: Xor immediate or register

---

Fall 2011  
EECS150 Lecture 8  
Page 41

---

Fall 2011  
EECS150 Lecture 8  
Page 42
Single-Cycle and Pipelined Datapath

Corrected Pipelined Datapath

- WriteReg must arrive at the same time as Result
Pipelined Control

Same control unit as single-cycle processor
Control delayed to proper pipeline stage

Pipeline Hazards

- Occurs when an instruction depends on results from previous instruction that hasn’t completed.
- Types of hazards:
  - **Data hazard**: register value not written back to register file yet
  - **Control hazard**: next instruction not decided yet (caused by branches)