Review of the Machine Cycle

- When a program is executing, its instructions are located in main memory. The address of an instruction is the address of the first (the lowest addressed) byte of the four-byte instruction.
- Each machine cycle executes one machine instruction. At the top of the machine cycle, the PC (program counter) contains the address of an instruction to fetch from memory. The instruction is fetched into the processor and is prepared for execution.
- In the middle of the machine cycle the PC is incremented by four so that it points to the instruction that follows the one just fetched. Then the fetched instruction is executed and the cycle repeats. The machine cycle automatically executes instructions in sequence.
- When a jump instruction executes (in the last step of the machine cycle), it puts a new address into the PC. Now the fetch at the top of the next machine cycle fetches the instruction at that new address. Instead of executing the instruction that follows the jump instruction in memory, the processor "jumps" to an instruction somewhere else in memory.
Review of the Machine Cycle

- However, it takes an extra machine cycle before the change in the PC takes effect. Before the PC changes, the instruction that follows the jump instruction in memory is fetched and executed. After that instruction executes, the next instruction to execute is the one that was jumped to. The instruction that follows a jump instruction in memory is said to be in the branch delay slot.
- The reason for this delay is that MIPS is pipelined. Normally instructions are executed in sequence. In order to gain speed, the processor cleverly fetches several sequential instructions and starts working on them all. When the machine cycle calls for one of these instructions to be executed, most of the work has already been done. These instructions are in an instruction pipe.
- This means that the instruction after a jump instruction has mostly been completed when the jump is executed. Rather than waste this effort, the instruction is allowed to finish. Only then is the PC changed by the jump instruction.
- The instruction that follows a jump instruction in memory (in the branch delay slot) is always executed. After it executes, the next instruction to execute is the one that was jumped to. Often the branch delay slot is filled with a nop instruction.
- The QTSpiM simulator allows you to turn the pipeline feature off, but this is not an option with actual R2000 hardware.

Altering the PC

- Shown below is a sequence of instructions. The "load" and "add" represent typical instructions. The "jump" instruction shows the address we wish to put into the PC. (The actual MIPS instruction for this involves details that we are skipping for the moment.)
- The last instruction, a nop, fills the branch delay slot to give the PC time to change. Once started, the four instructions execute in an unending loop.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction (details omitted)</th>
<th>PC just after this instruction has executed (at the bottom of the cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00400000</td>
<td>load</td>
<td>00400004</td>
</tr>
<tr>
<td>00400004</td>
<td>add</td>
<td>00400008</td>
</tr>
<tr>
<td>00400008</td>
<td>jump 0x00400000</td>
<td>0040000C</td>
</tr>
<tr>
<td>0040000C</td>
<td>nop</td>
<td>00400000 -- effect of jump</td>
</tr>
</tbody>
</table>

- A loop structure is created with the jump instruction. The intent of the jump instruction is to put the address 0x00400000 into the PC. However, this effect is not seen until after the instruction in the branch delay slot has executed.
Review of MIPS Instruction Formats

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point
    - Coprocessor
  - Memory Management
  - Special

- 3 Instruction Formats - all 32 bits wide

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>I format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>jump target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>J format</td>
</tr>
</tbody>
</table>

Addresses in Jump Instructions

- PC-relative addressing is used for jump instructions:

<table>
<thead>
<tr>
<th>J</th>
<th>op</th>
<th>26 bit address</th>
</tr>
</thead>
</table>

- The MIPS jump instruction (j) replaces the lower 28 bits of the PC with A00 where A is the 26 bit address; it never changes the upper 4 bits:
  - Example: if PC = 1011X (where X = 28 bits), it is replaced with 1011A00
  - There are 16 (=2^4) partitions of the 2^32 size address space, each partition of size 256 MB (=2^28), such that, in each partition the upper 4 bits of the address is the same.
  - If a program crosses an address partition, then a jump instruction that attempts to jump to a different partition has to be replaced by (jr) with a full 32-bit address first loaded into the jump register.
Unconditional Branch (Jump)

- MIPS unconditional branch instructions:
  \[ j \text{ Label} \]
- Example:
  \[
  \text{if (i!=j)} \quad \text{beq } s4, s5, \text{Lab1}
  \quad h=i+j;
  \quad \text{add } s3, s4, s5
  \quad \text{else}
  \quad j \text{ Lab2}
  \quad h=i-j;
  \quad \text{Lab1: sub } s3, s4, s5
  \quad \text{Lab2: } \ldots
  \]

<table>
<thead>
<tr>
<th>op</th>
<th>26 bit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>00000000000000000000011001</td>
</tr>
</tbody>
</table>

6 bits 26 bits

Addresses in Branch

- Instructions:
  \[
  \text{bne } t4,t5,\text{Label} \quad \text{Next instruction is at Label if } t4 \neq t5
  \quad \text{beq } t4,t5,\text{Label} \quad \text{Next instruction is at Label if } t4 = t5
  \]
- Format:
  \[
  \begin{array}{cccc}
    \text{op} & \text{rs} & \text{rt} & \text{16 bit offset} \\
  \end{array}
  \]
- 16 bits is too small to reach a \(2^{32}\) address space.
- Solution: specify a register (as for \text{lw} and \text{sw}) and add it to offset:
  - Use the Program Counter as the register. This type of addressing is called \(\text{PC-relative}\) addressing, based on:
    - \textit{Principle of locality}: most branches are to instructions near current instruction (e.g., loops and if statements).
Addresses in Branch

- We can further extend the reach of branch instructions by observing that all MIPS instructions are a word (= 4 bytes), therefore we use word-relative addressing.
- MIPS branch destination address = $(PC + 4) + (4 * offset)$

Because hardware typically increments PC early in execute cycle to point to next instruction

- So offset = (branch destination address – PC – 4)/4
- But QTSipm does offset = (branch destination address – PC)/4 if pipelining is not enabled so ….
  - Be careful, best to stick to branching to labels, rather than calculating the offset by hand.

MIPS Compare and Branch Instructions

- Compare and Branch
  - beq rs, rt, offset if $R[rs] == R[rt]$ then PC-relative branch
  - bne rs, rt, offset $<>$
- Compare to Zero and Branch
  - blez rs, offset if $R[rs] <= 0$ then PC-relative branch
  - bgtz rs, offset $>$
  - blt rs, offset $<$
  - bgez rs, offset $>=$
  - bltzal rs, offset if $R[rs] < 0$ then branch and link (into R31)
  - bgezal rs, offset $>=$
- Why would you guess that only compares with zero are supported and not general compares?
# MIPS Compare and Jump Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>set on less than</td>
<td><code>slt $1, $2, $3</code></td>
<td>if ((\text{\textless} 1 &lt; \text{\textless} 3)) (\text{\textless} 1) = 1; else (\text{\textless} 1) = 0) (\text{\textless} 1) = 2's complement</td>
</tr>
<tr>
<td>set less than imm.</td>
<td><code>slti $1, $2, 100</code></td>
<td>if ((\text{\textless} 1 &lt; 100)) (\text{\textless} 1) = 1; else (\text{\textless} 1) = 0) (\text{\textless} 1) = 2's complement</td>
</tr>
<tr>
<td>set less than uns.</td>
<td><code>sltu $1, $2, $3</code></td>
<td>if ((\text{\textless} 1 &lt; \text{\textless} 3)) (\text{\textless} 1) = 1; else (\text{\textless} 1) = 0) (\text{\textless} 1) = unsigned numbers</td>
</tr>
<tr>
<td>set l. t. imm. uns.</td>
<td><code>sltiu $1, $2, 100</code></td>
<td>if ((\text{\textless} 1 &lt; 100)) (\text{\textless} 1) = 1; else (\text{\textless} 1) = 0) (\text{\textless} 1) = unsigned numbers</td>
</tr>
<tr>
<td>jump</td>
<td><code>j 10000</code></td>
<td>go to (PC[31,28], 40000) (\text{\textless} 1) = jump to target address</td>
</tr>
<tr>
<td>jump and link</td>
<td><code>jal 10000</code></td>
<td>(\text{\textless} 1) = PC + 4; go to (PC[31,28], 40000) (\text{\textless} 1) = for procedure/subroutine call</td>
</tr>
<tr>
<td>jump register</td>
<td><code>jr $31</code></td>
<td>go to (\text{\textless} 1)) = for switch, procedure/subroutine return</td>
</tr>
</tbody>
</table>

## Signed vs. Unsigned Comparison

\begin{align*}
R1 &= 0\ldots00 0000 0000 0000 0001_{2} = 1_{10} \\
R2 &= 0\ldots00 0000 0000 0000 0010_{2} = 2_{10} \\
R3 &= 1\ldots11 1111 1111 1111 1111_{2} = 2^{31} - 1 \text{ or } -1_{10}
\end{align*}

- After executing these instructions:
  
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>slt</td>
<td><code>r4, r2, r1</code></td>
</tr>
<tr>
<td>slt</td>
<td><code>r5, r3, r1</code></td>
</tr>
<tr>
<td>sltu</td>
<td><code>r6, r2, r1</code></td>
</tr>
<tr>
<td>sltu</td>
<td><code>r7, r3, r1</code></td>
</tr>
</tbody>
</table>

- What are values of registers \(r4\) - \(r7\)?

\begin{align*}
\text{\textless} r4 \text{\textless} = &; \text{\textless} r5 \text{\textless} = &; \text{\textless} r6 \text{\textless} = &; \text{\textless} r7 \text{\textless} = &
\end{align*}
MIPS Code for If-then-else Structure

- Conditional statements allow us to make decisions.
- Replace the C code for
  
  ```
  if (i == j), f = g + h; else f = g - h;
  ```

  by equivalent MIPS instructions.
- Assume variables f through j correspond to registers $s0$ through $s4$.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>bne $s3, $s4, Else</td>
<td>if (i != j) goto Else</td>
</tr>
<tr>
<td>add $s0, $s1, $s2</td>
<td>f = g + h</td>
</tr>
<tr>
<td>j Exit</td>
<td>go to Exit</td>
</tr>
<tr>
<td>Else:</td>
<td></td>
</tr>
<tr>
<td>sub $s0, $s1, $s2</td>
<td>f = g - h</td>
</tr>
<tr>
<td>Exit:</td>
<td></td>
</tr>
</tbody>
</table>

Pseudo-Instructions

- The pseudo-instruction
  
  ```
  blt $s0, $s1, Else
  ```

  is implemented as
  
  ```
  slt $at, $s0, $s1
  bne $at, $zero, Else
  ```

- Check the course web page for other branch instructions and branching pseudo-instructions.
- Note that when some pseudo-instructions are “expanded” into actual MIPS instructions, $R1 is often used to store temporary values.
Example: BranchJump.asm

# Nonsense program to show address calculations for branch and jump instructions

.text  # text section
.globl main  # call main by SPIM

# Nonsense code - load in SPIM to see the address calculations
main:
    j label
    add $0, $0, $0
    beq $8, $9, label
    add $0, $0, $0
    add $0, $0, $0
    add $0, $0, $0
    add $0, $0, $0

label:
    add $0, $0, $0

Conclusions

• Program flow is controlled by branch and jump instructions;
• Decisions are made by comparing the contents of two registers or the contents of one register with an immediate value;
• Jump instructions use word-relative addressing and are limited to 256Mbyte partitions;
• Branch instructions use offsets and are limited to a 16 bit address space.