EECS 322: Computer Architecture

The SPIM simulator
Website and Homework

Homework Website (temporary)
http://sfo.ces.cwru.edu

Problems from book (427-432)
  5.1-2, 5.5-6, 5.9, 5.14-18, 5.24
Von Neuman & Harvard Architectures (PH p. 35)

Von Neuman architecture was coined to describe machines with separate memories. Speed efficient: Increased parallelism.

Harvard architecture was area efficient but requires higher bus bandwidth because instructions and data must compete for memory.
Recap: Single Cycle Implementation

- Calculate instruction cycle time assuming negligible delays except:
  - memory (2ns), ALU and adders (2ns), register file access (1ns)

Single Cycle = 2 adders + 1 ALU + 4 muxes
Multi-cycle Datapath

Multi-cycle = 5 Muxes + 1 ALU + Controller + 4 Registers (A,B,MDR,ALUOut)
Single-cycle = 4 Muxes + 1 ALU + 2 adders
Single/Multi-Clock Comparison

\[
\begin{align*}
\text{add} & = 6\text{ns} = \text{Fetch}(2\text{ns}) + \text{RegR}(1\text{ns}) + \text{ALU}(2\text{ns}) + \text{RegW}(2\text{ns}) \\
\text{lw} & = 8\text{ns} = \text{Fetch}(2\text{ns}) + \text{RegR}(1\text{ns}) + \text{ALU}(2\text{ns}) + \text{MemR}(2\text{ns}) + \text{RegW}(2\text{ns}) \\
\text{sw} & = 7\text{ns} = \text{Fetch}(2\text{ns}) + \text{RegR}(1\text{ns}) + \text{ALU}(2\text{ns}) + \text{MemW}(2\text{ns}) \\
\text{beq} & = 5\text{ns} = \text{Fetch}(2\text{ns}) + \text{RegR}(1\text{ns}) + \text{ALU}(2\text{ns}) \\
\text{j} & = 2\text{ns} = \text{Fetch}(2\text{ns})
\end{align*}
\]

\[
\frac{\text{CPU single\text{--}cycle\text{clock}}}{\text{CPU multi\text{--}cycle\text{clock}}} = \frac{8\text{ns}}{6.3\text{ns}} = 1.27 \text{ times faster}
\]

Architectural improved performance without speeding up the clock!
Microprogramming: program overview

T₁ → Fetch

T₂ → Fetch+1

T₃ → Dispatch 1
  → Rformat₁
  → BEQ₁
  → JUMP₁
  → Mem₁

T₄ → Dispatch 2
  → Rformat₁+1
  → LW₁
  → SW₁

T₅ → LW₁+1
The Spim Simulator


Spim documentation
Appendix A.9 SPIM Patterson & Hennessy pages A-38 to A75

Spim runnable code samples (Hello_World.s, simplecalc.s, ...)
  http://vip.cs.utsa.edu/classes/cs2734s98/overview.html

Other useful links
http://www.cs.wisc.edu/~larus/spim.html
# MIPS registers and conventions

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Conventional usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>0</td>
<td>Constant 0</td>
</tr>
<tr>
<td>$v0-$v1</td>
<td>2-3</td>
<td>Expression evaluation &amp; function results</td>
</tr>
<tr>
<td>$a0-$a3</td>
<td>4-7</td>
<td>Arguments 1 to 4</td>
</tr>
<tr>
<td>$t1-$t9</td>
<td>8-15,24,35</td>
<td>Temporary (not preserved across call)</td>
</tr>
<tr>
<td>$s0-$s7</td>
<td>16-23</td>
<td>Saved Temporary (preserved across call)</td>
</tr>
<tr>
<td>$k0-$k1</td>
<td>26-27</td>
<td>Reserved for OS kernel</td>
</tr>
<tr>
<td>$gp</td>
<td>28</td>
<td>Pointer to global area</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>$fp</td>
<td>30</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>Return address (used by function call)</td>
</tr>
</tbody>
</table>
MIPS Register Name translation

# calculate f = (g + h) - (i + j)  (PH p. 109, file: simplecalc.s)

<table>
<thead>
<tr>
<th>Assembler .s</th>
<th>Translated (1 to 1 mapping)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addi $s1, $0, 5</td>
<td>addi $17, $0, 5 #g = 5</td>
</tr>
<tr>
<td>addi $s2, $0, -20</td>
<td>addi $18, $0, -20 #h = -20</td>
</tr>
<tr>
<td>addi $s3, $0, 13</td>
<td>addi $19, $0, -20 #i = 13</td>
</tr>
<tr>
<td>addi $s4, $0, 3</td>
<td>addi $20, $0, 3 #j = 3</td>
</tr>
<tr>
<td>add $t0, $s1, $s2</td>
<td>add $8, $17, $18 #$t0=g + h</td>
</tr>
<tr>
<td>add $t1, $s3, $s4</td>
<td>add $9, $19, $20 #$t1=i + j</td>
</tr>
<tr>
<td>sub $s0, $t0, $t1</td>
<td>sub $16, $8, $9 #f=(g+h)-(i+j)</td>
</tr>
</tbody>
</table>
System call 1: print_int $a0

• System calls are used to interface with the operating system to provide device independent services.

• System call 1 converts the binary value in register $a0 into ascii and displays it on the console.

• This is equivalent in the C Language: printf("%d", $a0)

Assember .s

li       $v0, 1
add       $a0,$0,$s0
syscall

Translated (1 to 1 mapping)

ori     $2, $0, 1    #print_int (system call 1)
add    $4,$0,$16    #put value to print in $a0
syscall
## System Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Code</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_int</td>
<td>1</td>
<td>$a0=integer</td>
<td></td>
</tr>
<tr>
<td>print_float</td>
<td>2</td>
<td>$f12= float</td>
<td></td>
</tr>
<tr>
<td>print_double</td>
<td>3</td>
<td>$f12=double</td>
<td></td>
</tr>
<tr>
<td>print_string</td>
<td>4</td>
<td>$a0=string</td>
<td></td>
</tr>
<tr>
<td>read_int</td>
<td>5</td>
<td></td>
<td>$v0=integer</td>
</tr>
<tr>
<td>read_float</td>
<td>6</td>
<td></td>
<td>$f0= float</td>
</tr>
<tr>
<td>read_double</td>
<td>7</td>
<td></td>
<td>$f0= double</td>
</tr>
<tr>
<td>read_string</td>
<td>8</td>
<td>$a0=buf, $a1=len</td>
<td></td>
</tr>
<tr>
<td>sbrk</td>
<td>9</td>
<td>$a0=amount</td>
<td>$v0=address</td>
</tr>
<tr>
<td>exit</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
System call 4: print_string $a0

- System call 4 copies the contents of memory located at $a0 to the console until a zero is encountered.
- This is equivalent in the C Language: `printf("%s", $a0)`

Assembler .s
```
.data
.globl msg3
msg3: .asciiz "\nThe value of f is: "
.text
li $v0, 4 ori $2, $0, 4    #print_string
la $a0, msg3 lui $4,4097     #address of string
syscall syscall
```

Translated (1 to 1 mapping)
```
msg3 is just a label but must match
ori $2, $0, 4    #print_string
li $4,4097       #address of string
syscall
```
.asciiz data representations

.data: items are place in the data segment
    which is not the same as the .text segment!

Assembler .s
msg3:    .asciiz "\nThe va"

Same as in assembler.s
msg3:    .byte  \n',T',h', e', ', v', a', 0

Same as in assembler.s
msg3:    .byte  0x0a, 0x54, 0x68, 0x65
          .byte  0x20, 0x76, 0x61, 0x00

Same as in assembler.s
msg3:    .word 0x6568540a, 0x00617620

Translated in the .data segment: 0x6568540a 0x00617620

Big endian format
Memory layout: segments

- Segments allow the operating system to protect memory
- Like Unix file systems: .text Execute only, .data R/W only

```
addi $17,0,5
addi $18,0,-20

.ascii "The value of f is "
```
Hello, World: hello.s

```c
#include <stdio.h>

int main() {
    printf("Hello World\n");
    return 0;
}
```

```
GLOBAL main

main:
    #main has to be a global label
    addu $s7, $0, $ra #save the return address in a global reg.

GLOBAL hello

hello: .asciiz "Hello World\n" #string to print

.text
    li $v0, 4 # print_str (system call 4)
    la $a0, hello # $a0=address of hello string
    syscall

# Usual stuff at the end of the main
    addu $ra, $0, $s7 #restore the return address
    jr $ra #return to the main program
    add $0, $0, $0 #nop
```
Simplecalc.s (PH p. 109)

.globl main
main:
  addu $s7, $0, $ra  # save the return address
  addi $s1, $0, 5    # g = 5
  addi $s2, $0, -20  # h = -20
  addi $s3, $0, 13   # i = 13
  addi $s4, $0, 3    # j = 3
  add $t0, $s1, $s2  # register $t0 contains g + h
  add $t1, $s3, $s4  # register $t1 contains i + j
  sub $s0, $t0, $t1  # f = (g + h) - (i + j)
  li $v0, 4          # print_str (system call 4)
  la $a0, message    # address of string
  syscall
  li $v0, 1          # print_int (system call 1)
  add $a0, $0, $s0   # put value to print in $a0
  syscall
  addu $ra, $0, $s7  # restore the return address
  jr $ra             # return to the main program
  add $0, $0, $0     # nop

.data
.message:       .asciiz "\nThe value of f is: "  # string to print
.globl message
Order of .text and .data not important
### Simplecalc.s without symbols (PH p. 109)

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00400020</td>
<td>addu $23, $0, $31</td>
<td># addu $s7, $0, $ra</td>
</tr>
<tr>
<td>0x00400024</td>
<td>addi $17, $0, 5</td>
<td># addi $s1, $0, 5</td>
</tr>
<tr>
<td>0x00400028</td>
<td>addi $18, $0, -20</td>
<td># addi $s2, $0, -20</td>
</tr>
<tr>
<td>0x0040002c</td>
<td>addi $19, $0, 13</td>
<td># addi $s3, $0, 13</td>
</tr>
<tr>
<td>0x00400030</td>
<td>addi $20, $0, 3</td>
<td># addi $s4, $0, 3</td>
</tr>
<tr>
<td>0x00400034</td>
<td>add $8, $17, $18</td>
<td># add $t0, $s1, $s2</td>
</tr>
<tr>
<td>0x00400038</td>
<td>add $9, $19, $20</td>
<td># add $t1, $s3, $s4</td>
</tr>
<tr>
<td>0x0040003c</td>
<td>sub $16, $8, $9</td>
<td># sub $s0, $t0, $t1</td>
</tr>
<tr>
<td>0x00400040</td>
<td>ori $2, 0, 4</td>
<td>#print_str (system call 4)</td>
</tr>
<tr>
<td>0x00400044</td>
<td>lui $4, 0x10010000</td>
<td># address of string</td>
</tr>
<tr>
<td>0x00400048</td>
<td>syscall</td>
<td></td>
</tr>
<tr>
<td>0x0040004c</td>
<td>ori $2, 1</td>
<td>#print_int (system call 1)</td>
</tr>
<tr>
<td>0x00400050</td>
<td>add $4, $0, $16</td>
<td>#put value to print in $a0</td>
</tr>
<tr>
<td>0x00400054</td>
<td>syscall</td>
<td></td>
</tr>
<tr>
<td>0x00400058</td>
<td>addu $31, $0, $23</td>
<td>#restore the return address</td>
</tr>
<tr>
<td>0x0040005c</td>
<td>jr $31</td>
<td>#return to the main program</td>
</tr>
<tr>
<td>0x00400060</td>
<td>add $0, $0, $0</td>
<td>#nop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>.data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10010000</td>
<td>.word 0x6568540a, 0x6c617620, 0xf206575</td>
</tr>
<tr>
<td></td>
<td>.word 0x20662066, 0x203a7369, 0x00000000</td>
</tr>
</tbody>
</table>
## Single Stepping

<table>
<thead>
<tr>
<th></th>
<th>$pc</th>
<th>$t0</th>
<th>$t1</th>
<th>$s0</th>
<th>$s1</th>
<th>$s2</th>
<th>$s3</th>
<th>$s4</th>
<th>$s7</th>
<th>$ra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00400030</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>5</td>
<td>ffffffffc</td>
<td>0d</td>
<td>?</td>
<td>400018</td>
<td>400018</td>
</tr>
<tr>
<td></td>
<td>00400034</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>5</td>
<td>ffffffffc</td>
<td>0d</td>
<td>3</td>
<td>400018</td>
<td>400018</td>
</tr>
<tr>
<td></td>
<td>00400038</td>
<td>ffffffff1</td>
<td>?</td>
<td>?</td>
<td>5</td>
<td>ffffffffc</td>
<td>0d</td>
<td>?</td>
<td>400018</td>
<td>400018</td>
</tr>
<tr>
<td></td>
<td>0040003c</td>
<td>?</td>
<td>10</td>
<td>?</td>
<td>5</td>
<td>ffffffffc</td>
<td>0d</td>
<td>?</td>
<td>400018</td>
<td>400018</td>
</tr>
<tr>
<td></td>
<td>00400040</td>
<td>?</td>
<td>?</td>
<td>ffffffffe1</td>
<td>5</td>
<td>ffffffffc</td>
<td>0d</td>
<td>?</td>
<td>400018</td>
<td>400018</td>
</tr>
</tbody>
</table>