MIPS Functions/Subroutines

Gabriel Hugh Elkaim
MSI Rescheduled

Friday - Account of McNary
Room 332
11 am - Noon
Functions/Subroutines (1.3)

\[ x = \text{average} (9, 3); \]
Functions/Subroutines (2.3)
Functions/Subroutines (3.3)

ra gets overwritten every time I call jal
Function Example: Count Chars

"The world is round!"

count characters

$\texttt{\$ae}$ - character I'm interested in

$\texttt{\$ai}$ - string address

$\texttt{\$vo}$ - count
Function Example: Find First Char

Find first character

\$x0 \text{ character}

\$x1 \text{ string address}

\$x0 \text{ addru of first char or 10.}
Function Example: CountChars

The bear is in the woods.

```
jal FindFirstChar

d40 = main[vphi]

isphi = 0?

kf:
  count = count + 1
  count = count + 1

return count
```
Function Example: FindFirstChar

$t\phi = a1$

$t1 = \text{men}(t\phi)$

$t\phi = t\phi + 1$

$t1 = a1$

\[t1 = 0\]  
Yes: return $t\phi$

No:

\[t1 = a1\]  
Yes: return $t\phi$

No: return 'e'

The bear ↑↑ 'e'
MIPS Instruction: NOP

$\text{NOP} = \text{no operation}$
Macros (1.3)

copy and paste

li $v0, 10
syscall

.macro Exit
li $v0, 10
syscall
.end_macro
.macro PrintInt( %x )
    li $v0, 1
    add $a0, $t1, $3200
    syscall
    end macro

PrintInt( 154 )
PrintInt( 33 )
Macros (3.3)
MARS: .eqv

.eqv - equivalent

.name replace

.eqv $count, $sf

.addi $t1, $s5, 1

.addi, $t candid, $s candid, 1
MIPS Instruction Decoding

Gabriel Hugh Elkaim
## Register Conventions

<table>
<thead>
<tr>
<th>Register Number</th>
<th>Conventional Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>$zero</td>
<td>Hard-wired to 0</td>
</tr>
<tr>
<td>$1</td>
<td>$at</td>
<td>Reserved for pseudo-instructions</td>
</tr>
<tr>
<td>$2 - $3</td>
<td>$v0, $v1</td>
<td>Return values from functions</td>
</tr>
<tr>
<td>$4 - $7</td>
<td>$a0 - $a3</td>
<td>Arguments to functions - not preserved by subprograms</td>
</tr>
<tr>
<td>$8 - $15</td>
<td>$t0 - $t7</td>
<td>Temporary data, not preserved by subprograms</td>
</tr>
<tr>
<td>$16 - $23</td>
<td>$s0 - $s7</td>
<td>Saved registers, preserved by subprograms</td>
</tr>
<tr>
<td>$24 - $25</td>
<td>$t8 - $t9</td>
<td>More temporary registers, not preserved by subprograms</td>
</tr>
<tr>
<td>$26 - $27</td>
<td>$k0 - $k1</td>
<td>Reserved for kernel. Do not use.</td>
</tr>
<tr>
<td>$28</td>
<td>$gp</td>
<td>Global Area Pointer (base of global data segment)</td>
</tr>
<tr>
<td>$29</td>
<td>$sp</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>$30</td>
<td>$fp</td>
<td>Frame Pointer</td>
</tr>
<tr>
<td>$31</td>
<td>$ra</td>
<td>Return Address</td>
</tr>
</tbody>
</table>
Program Counter
R-type Instructions

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>rs</th>
<th>rs</th>
<th>rd</th>
<th>SHamt</th>
<th>Func</th>
</tr>
</thead>
<tbody>
<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>
</tr>
</tbody>
</table>

$\text{adr, add, str}$
R-type: ADD

```
ADD $t0, $t1, $t2
```

**Function:**

```
0x20
```

**Binary Format:**

```
000000 01001 01010 010000 00000 100000
```

- **opcode:** 000000
- **rs:** 01001
- **rt:** 01010
- **rd:** 01000
- **shamt:** 00000
- **funct:** 100000

**Characteristics:**

```
0x012A9020
```

---

*Gabriel Hugh Elkaim*
R-type: AND

\[ \text{OP code} \quad t_1 \quad t_2 \quad b_0 \]

\[ 000000 \]

\[ b_0 \quad b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5 \]

\[ \text{0x24} \]

\[ 100100 \]
R-type: MULT

```
00 0000  R3  R2  R0  Sdata  FUNC 0x18

01 1000
```
I-type Instruction

```
addi t0, t1, 48
```

<table>
<thead>
<tr>
<th>op code</th>
<th>rs</th>
<th>rt</th>
<th>constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16 bits</td>
</tr>
</tbody>
</table>
I-type Instruction
I-type: ADDI

addi $t0, $t1, 1

001000  01000  01001  0000000000000001

op code  rs  rd  Imm.

0x21090001

Im
I-type: LA (LUI/ORI)

.lh t0, $0o

.text
la t6, $0o

.data
$s0: .word -1

$lui $1, 0x0001
ori t8, $1, 0

0x0000 0000

→ gp
J-type Instruction

- **Op Code**: 6 bits
- **Target**: 26 bits

Last two bits will be zero

\[ <2000 \]
Branches

\[
\text{op code} \quad \frac{rs}{r} \quad \frac{rt}{r} \quad \text{addmiss} \quad \text{sx bits} \quad l \quad c
\]
PC-Relative Addressing (1.3)

locality of reference

most programs, code is near by
PC-Relative Addressing (2.3)

pc ← + 6 words

0 011000

∥

Seq

 Lag
PC-Relative Addressing (3.3)

\[ \text{beg } t0, t1, \text{ label} \]

\[ \text{beg } t0, t1, \text{ pc_offset} \]

\[ \text{bar: beg } \$0, \$0, \text{ bar} \]

\[ \text{beg } \$0, \$0, -1 \]
MIPS Assembly To Machine

Gabriel Hugh Elkaim
MIPS Program (Encryption)

.text
la $t0, Hello

# comments don't exist to the system
startL: lb $t2,0($t0)
beq $t2,$zero,endL
xori $t2,$t2,0xAA
sb $t2,0($t0)
addi $t0,$t0,1
b startL

data
endL:
li $v0, 10 # 10 is the exit syscall.
syscall # do the syscall.

.data
Hello: .asciiz "String to Garble"
<table>
<thead>
<tr>
<th>Address</th>
<th>Converted Assembly</th>
<th>Hex Machine Code</th>
<th>Binary Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00400000</td>
<td>lui $1, 0x00001001</td>
<td>0x3C011001</td>
<td>00111100000000100010000000000001</td>
</tr>
<tr>
<td>0x00400004</td>
<td>ori $8, $1, 0x0</td>
<td>0x34280000</td>
<td>00110100010100000000000000000000</td>
</tr>
<tr>
<td>0x00400008</td>
<td>lb $10, 0x0 ($8)</td>
<td>0x8110A000</td>
<td>100000100010000010100000000000000</td>
</tr>
<tr>
<td>0x0040000C</td>
<td>beq $10, $0, 0x4</td>
<td>0x11400004</td>
<td>00010001010000000000000000000000</td>
</tr>
<tr>
<td>0x00400010</td>
<td>xori $10, $10, 0xAA</td>
<td>0x39A000AA</td>
<td>00111001101000000000000000000010</td>
</tr>
<tr>
<td>0x00400014</td>
<td>sb $10, 0x0 ($8)</td>
<td>0xA10A000</td>
<td>10100001000010100000000000000000</td>
</tr>
<tr>
<td>0x00400018</td>
<td>addi $8, $8, 0x1</td>
<td>0x21080001</td>
<td>00100001000010000000000000000001</td>
</tr>
<tr>
<td>0x0040001C</td>
<td>bgez $0, 0xFFFFFFFFA</td>
<td>0x0401FFFF</td>
<td>00000100000000001111111111111101</td>
</tr>
<tr>
<td>0x00400020</td>
<td>addiu $2, $0, 0xA</td>
<td>0x2402000A</td>
<td>00100100000000000000000000000010</td>
</tr>
<tr>
<td>0x00400024</td>
<td>syscall</td>
<td>0x0000000C</td>
<td>00000000000000000000000000000010</td>
</tr>
</tbody>
</table>
Instruction Processing

1. Fetch instruction from memory
2. Decode instruction
3. Evaluate address
4. Fetch operands from memory
5. Execute operation
6. Store result
Instruction Decoding

- PC
- INSTR
- INSTM
- (hexA)
Register Block and ALU

[Diagram of register block and ALU with labels and connections]
Data Memory and Sign Extension

- Address: 32-bit input
- Data to write: 32-bit input
- Mem write: 32-bit output
- Mem read: 32-bit output

- SE: Sign extension block
- 16-bit input
- 32-bit output
Exceptions and Traps (1.4)

- Internal Condition
  - syscall
  - illegal instructions
  - arithmetic overflow
  - lw/sw unaligned

*Die Ghassouny.*
Exceptions and Traps (2.4)
Exceptions and Traps (3.4)

- Synchronously
- Redo it again if you rerun code
- Priorities
  - Must be handled immediately
  - NMI
Exceptions and Traps (4.4)

lw $t0, 1($fp)

PC → FPC
disable handler code
Jump to exception handler

EPC → PC
resume code
exception handler

Save AT
Save h0, h1

Save(A) is oword
Save A1 is oword

determine cause
Call appropriate handler

— cause

NO STACK!!
Multiple Exceptions

- Handle highest Priority first
- Equal (arbitrary)
  - exception within an exception
    1) Wait
    2) Exception re-entrant
I/O – Input and Output

behavior - how do I access

data type - how much data do I send/receive
I/O Controller

control register
status register

[Diagram showing CMOS/STATUS and DATA with an arrow pointing to Electronics]
Memory Mapped I/O

Special opcodes:

- `OP PrintCurr`

Hardware support limited.
Transfer Timing

**Synchronous**

- Data at fixed rate
- CPU runs every n cycles

**Asynchronous**

- Not predictable
- CPU must synchronize
- I/O status register
Transfer Control

Polling — CPU keeps checking status
"Are we there yet?"

Interrupt — Device sends a signal to CPU
- CPU is free
"Wake me up when we arrive"
Interrupts (1.4)
Interrupts (2.4)
Interrupts (3.4)
Interrupts (4.4)