EE 109 Unit 15

Subroutines

Stacks

Review of Program Counter

- PC is used to fetch an instruction
  - PC contains the address of the ________________
  - The value in the PC is placed on the address bus and the memory is told to read
  - The PC is incremented, and the process is repeated for the next instruction

GPR's Used for Subroutine Support

<table>
<thead>
<tr>
<th>Assembler Name</th>
<th>Reg. Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>$0</td>
<td>Constant 0 value</td>
</tr>
<tr>
<td>$at</td>
<td>$1</td>
<td>Assembler temporary</td>
</tr>
<tr>
<td>$v0-$v1</td>
<td>$2-$3</td>
<td>Procedure return values or expression evaluation</td>
</tr>
<tr>
<td>$a0-$a3</td>
<td>$4-$7</td>
<td>Arguments/parameters</td>
</tr>
<tr>
<td>$t0-$t17</td>
<td>$8-$15</td>
<td>Temporaries</td>
</tr>
<tr>
<td>$s0-$s7</td>
<td>$16-$23</td>
<td>Saved Temporaries</td>
</tr>
<tr>
<td>$t8-$t9</td>
<td>$24-$25</td>
<td>Temporaries</td>
</tr>
<tr>
<td>$a0-$a1</td>
<td>$26-$27</td>
<td>Reserved for OS kernel</td>
</tr>
<tr>
<td>$gp</td>
<td>$28</td>
<td>Global Pointer (Global and static variables/data)</td>
</tr>
<tr>
<td>$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 for current procedure</td>
</tr>
</tbody>
</table>
Subroutines (Functions)

- Subroutines are portions of code that we can call from anywhere in our code, execute that subroutine, and then ________________

```c
void main() {
    ...
    x = 8;
    res = avg(x, 4);
    ...
}
int avg(int a, int b){
    return (a+b)/2;
}
```

C code:

A subroutine to calculate the average of 2 numbers

We call the subroutine to calculate the average and return to where we called it

Subroutines

- Subroutines are similar to _____________ where we jump to a new location in the code

```c
void main() {
    ...
    x = 8;
    res = avg(x, 4);
    ...
}
int avg(int a, int b){
    return (a+b)/2;
}
```

C code:

Call "avg" sub-routine will require us to branch to that code

Normal Branches vs. Subroutines

- Difference between normal branches and subroutines branches is that with subroutines we have to return to where we left off
- We need to leave a ___________ to the return location ___________ we jump to the subroutine...once in the function its _____________

```c
void main() {
    ...
    x = 8;
    res = avg(x, 4);
    ...
}
int avg(int a, int b){
    return (a+b)/2;
}
```

C code:

Call "avg" sub-routine will require us to branch to that code

Implementing Subroutines

- To implement subroutines in assembly we need to be able to:
  - ________________ to the subroutine code
  - ________________ to when we finish the subroutine

```asm
.text
... 
jal  AVG ...
AVG: ...
jr   $ra
```

Assembly:
Jumping to a Subroutine

- **JAL instruction (Jump And Link)**
  - Format: `jal Address/Label`
  - Similar to jump where we load an address into the PC [e.g. \[PC = addr]\]
    - Same limitations (26-bit address) as jump instruction
    - Addr is usually specified by a label
- **JALR instruction (Jump And Link Register)**
  - Format: `jalr $rs`
  - Jumps to address specified by $rs (so we can jump a full 32-bits)
- In addition to jumping, JAL/JALR ____________ to be used as a link to return to after the subroutine completes

Returning from a Subroutine

- Use a JR with the $ra register to return to the instruction after the JAL that called this subroutine

Return Addresses

- No single return address for a subroutine since AVG may be called many times from many places in the code
- JAL always stores the address of the instruction after it (i.e. PC of ‘jal’ + 4)
Return Addresses

• A further complication is ________________
  ________________

• Example: Main routine calls SUB1 which calls SUB2

• Must store both return addresses but ________
  ________________

Dealing with Return Addresses

• Multiple return addresses can be spilled to memory
  — “Always” have enough memory

• Note: Return addresses will be accessed in ________
  _______ as they are stored
  — 0x400208 is the ________ RA to be stored but should be
    the ________ one used to return
  — A ________ is appropriate!

Stacks

• Stack is a data structure where data is accessed in reverse order as it is stored
  (a.k.a. LIFO = ________________)

• Use a stack to store the return addresses and other data

• System stack defined as growing towards ________ addresses
  — MARS starts stack at 0x7ffeffc
  — Normal MIPS starts stack at 0x80000000

• Top of stack is accessed and maintained using $sp=R[29] (stack pointer)
  — $sp points at top ________ location of the stack

Stack grows towards
  ________ addresses

2 Operations on stack

— ________: Put new data on top of stack
  • Decrement $sp
  • Write value to where $sp points

— ________: Retrieves and “removes” data from top of stack
  • Read value from where $sp points
  • Increment $sp to effectively “delete” top value

Push will add a value to the top of the stack

Pop will remove the top value from the stack
Push Operation

- Recall we assume $sp points at top occupied location
- Push: Put new data on top of stack
  - Decrement SP
    - Always decrement by 4 since addresses are always stored as words (32-bits)
  - Write return address ($ra) to where SP points
  - ___________________
- Always decrement by 4 since addresses are always stored as words (32-bits)

Push return address (e.g. 0x00400208)

Pop Operation

- Pop: Retrieves and "removes" data from top of stack
  - Read value from where SP points
    - ___________________
  - Increment SP to effectively "deletes" top value
    - Always increment by 4 when popping addresses

Subroutines and the Stack

- When writing native assembly, programmer must add code to manage return addresses and the stack
- At the beginning of a routine (PREAMBLE)
  - Push $ra (produced by 'jal') onto the stack
    - addi _____________
    - sw _____________
- Execute subroutine which can now freely call other routines
- At the end of a routine (POSTAMBLE)
  - Pop/restore $ra from the stack
    - lw _____________
    - addi _____________
    - jr $ra

Warning: Because the stack grows towards lower addresses, when you push something on the stack you subtract 4 from the SP and when you pop, you add 4 to the SP.
Optimizations for Subroutines

• Definition:
  – Leaf procedure: A procedure that ____________
  ____________________________________________

• Optimization
  – A leaf procedure need not save $ra onto the stack since it will not call another routine (and thus not overwrite $ra)

Leaf Subroutine

```
void main() {
    int arg1, arg2;
    ans = avg(arg1, arg2);
}

int avg(int a, int b) {
    int temp=1; // local var's
    return a+b >> temp;
}
```

Arguments and Return Values

• Most subroutine calls pass arguments/parameters to the routine and the routine produces return values
• To implement this, there must be locations agreed upon by caller and callee for where this information will be found
• MIPS convention is to use certain registers for this task
  – ________________ used to pass up to 4 arguments
  – ________________ used to return up to a 64-bit value

• Up to 4 arguments can be passed in $a0-$a3
  – If more arguments, use the stack
• Return value (usually HLL’s) limit you to one return value in $v0
  – For a 64-bit return value, use $v1 as well

Arguments and Return Values