Computer Architecture

Discussion 4

CB

SYSCALL

What is SYSCALL?

How SYSCALL works?

MIPS Interrupt

- Interrupts are events that demand the processor's attention
- Must be handled without affecting any active programs.
- Since interrupts can happen at any time, there is no way for the active programs to prepare for the interrupt.

When an Interrupt occurs,…

- **When an interrupt occurs, your processor may perform the following actions:**
- move the current PC into another register, call the EPC
- record the reason for the exception in the Cause register
- automatically disable further interrupts from occurring, by left-shifting the Status register
- change control (jump) to a hardwired interrupt handler address
- **To return from a handler, your processor may perform the following actions:**
- move the contents of the EPC register to the PC.
- re-enable interrupts, by right-shifting the Status register

Cause register

The Cause register is a 32-bit register, but only certain fields on that register will be used.
Bits 1 down to 0 will be set to describe the cause of the last interrupt/exception.

Number	Name	Description
00	INT	
01	IBUS	Instruction bus error (invalid instruction)
10	OVF	Arithmetic overflow
11	SYSCALL	System call

Status register

- The status register is also a 32-bit register.
- Bits 3 down to 0 will define masks for the three types of interrupts/exceptions.
- If an interrupt/exception occurs when its mask bit is current set to 0, then the interrupt/exception will be ignored.

Bit	Interrupt/exception
3	INT
2	IBUS
1	OVF
0	SYSCALL

SYSCALL in MARS

- Step 1. Load the service number in register $v0.
- Step 2. Load argument values, if any, in $a0, $a1, $a2, or $f12 as specified.
- Step 3. Issue the SYSCALL instruction.
- Step 4. Retrieve return values, if any, from result registers as specified.

Service	Code in $v0	Arguments	Result
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	……

Read and print an integer in MARS

- li $v0, 5
- Syscall
- add $t0,$v0,$zero
- li $v0, 1
- add $a0, $t0, $zero
- syscall

Exercise

- Input a string and output the substring that begins with the second character.

```
.data
STRING: .word 0:10
.text
li $v0,8
la $a0,STRING
li $a1,30
syscall
li $v0,4
la $a0,STRING
add $a0,$a0,1
syscall
```

SP

Using the Stack (2/2)

- Hand-compile

```
int sumSquare(int x, int y) {
    return mult(x,x)+ y; }
```

```
sumSquare:
         addi  $sp,$sp,-8     # space on stack
         sw    $ra, 4($sp)    # save ret addr
"push"   sw    $a1, 0($sp)    # save y
         add   $a1,$a0,$zero  # mult(x,x)
         jal   mult           # call mult
         lw    $a1, 0($sp)    # restore y
         add   $v0,$v0,$a1    # mult()+y
         lw    $ra, 4($sp)    # get ret addr
         addi  $sp,$sp,8      # restore stack
"pop"    jr    $ra
mult:    ...
```

Recursive Function Factorial

```
int fact (int n)
{
  if (n < 1) return (1);
     else return (n * fact(n-1));
}
```

Recursive Function Factorial

```
Fact:
   # adjust stack for 2 items
   addi $sp,$sp,-8
   # save return address
   sw $ra, 4($sp)
   # save argument n
   sw $a0, 0($sp)
   # test for n < 1
   slti $t0,$a0,1
   # if n >= 1, go to L1
   beq $t0,$zero,L1
   # Then part (n==1) return 1
   addi $v0,$zero,1
   # pop 2 items off stack
   addi $sp,$sp,8
   # return to caller
   jr $ra
```

```
L1:
   # Else part (n >= 1)
   # arg. gets (n - 1)
   addi $a0,$a0,-1
   # call fact with (n - 1)
   jal Fact
   # return from jal: restore n
   lw $a0, 0($sp)
   # restore return address
   lw $ra, 4($sp)
   # adjust sp to pop 2 items
   addi $sp, $sp,8
   # return n * fact (n - 1)
   mul $v0,$a0,$v0
   # return to the caller
   jr $ra
```

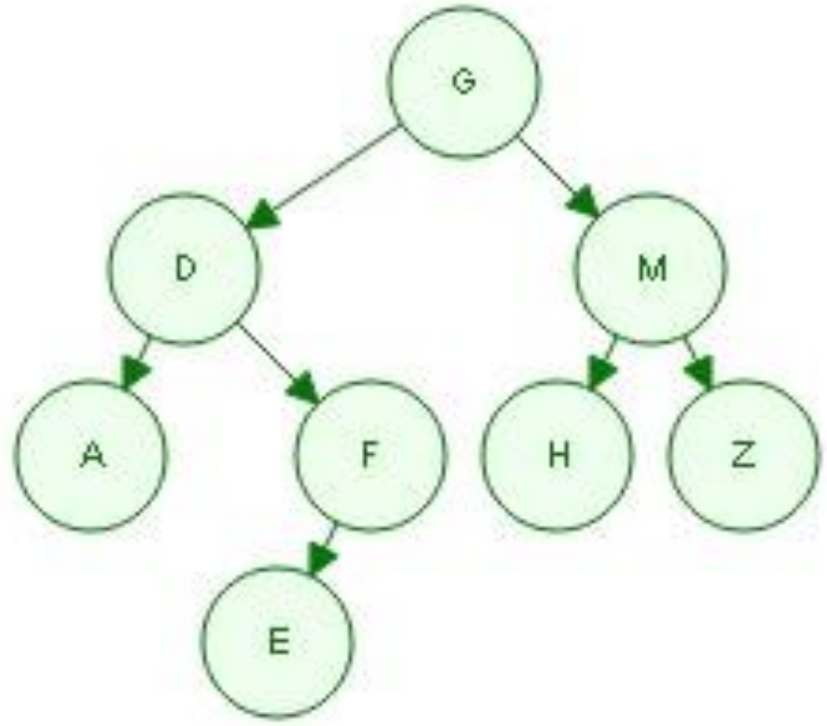
Exercise

```
Int fib(int n){
	if (n<2)
		return 1;
	return fib(n-1)+fib(n-2);
}
```

```
# int fib (int n)
fib:
    subu    sp, sp, 32  # Allocate a 32-byte stack frame
    sw  ra, 20(sp)  # Save Return Address
    sw  fp, 16(sp)  # Save old frame pointer
    addiu   fp, sp, 28  # Setup new frame pointer
    sw  a0,  0(fp)  # Save argument (n) to stack

    lw  v0, 0(fp)   # Load n into v0
    slti t0,v0,2      # if v0<2 ->1 else 0
    blez    t0, L2      # if t0 = 0 jump to rest of the function
    li  v0, 1       # n==1,0, return 1
    j   L1      # jump to frame clean-up code

L2:
    lw  v1, 0(fp)   # Load n into v1
    subu    v0, v1, 1   # Compute n-1
    move    a0, v0      # Move n-1 into first argument
    jal fib        # Recursive call
    move     t0, v0   # t0=v0

    lw  v1, 0(fp)   # Load n into v1
    subu    v0, v1, 2    # Compute n-2
    move     a0, v0    #  a0=n-2
    jal fib
    add v0, v0, t0  # Compute fib(n-1) + fib(n-2)

    #Result is in v0, so clean up the stack and return
L1:
    lw  ra, 20(sp)  # Restore return address
    lw  fp, 16(sp)  # Restore frame pointer
    addiu   sp, sp, 32  # Pop stack
    jr  ra      # return
    .end    fib
```

HW3 is out~ 2333

- Preorder NLR
- Inorder LNR
- Postorder LRN



- Given Inorder & Postorder, find Preorder.