17 MIPS assembly

Wednesday, October 14, 2015 10:01

Countdown.s

```
        .text
main:
        Li $s0, 10

        Loop:
li $v0, 1 # print_int
        move $a0, hello
        Syscall
        Li $v0, 4 #print)string
        La $a0, lf
        syscall

        sub $s0, $s0, 1 # assembler still knows subi
        Slt $s1, $s0,$ 0 # set t1 to result  of "t0 < 0"
        Beq $s1, $0, loop

        #li $v0, 10 # exit
        #syscall
        Jr $ra

        .data
lf:
        .asciiz "\n"
```

Subroutines

When we step, we see here are a couple steps before the actual program we wrote

Startup code

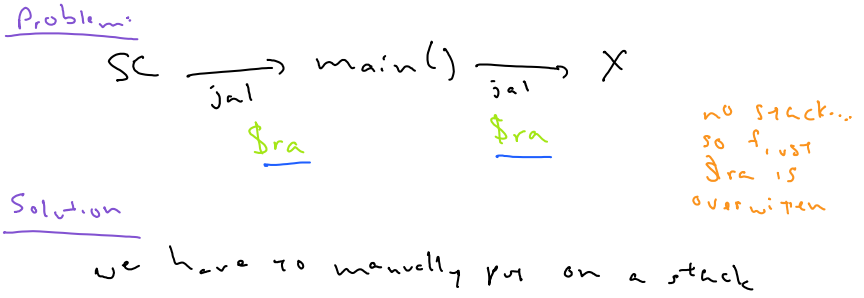
- Not just the simulator
- Actually happens
- Stuff before the main function
- In Java, it's the Virtual Machine
- Sc -> main()
    - Sc jal's to main
- Jal - jump and link
    - Jsr equivalent from 6502
    - Y jal x
    - Remembers the return address
        - Puts it in a register $31 or $ra
        - $ra <- y + 4
            - Because instruction is 4 bytes
    - Sets the PC <- x

main (

Note: no sub instruction

- No RTS instruction

spim uses addi neg.



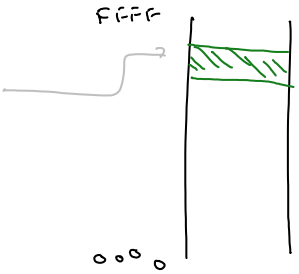
Cd2.s

```
        .text
main:
        Li $s0, 10

Loop:
        move $a0, $s0
        Jal printint

        sub $s0, $s0, 1 # assembler still knows subi
        Slt $s1, $s0,$ 0 # set t1 to result  of "t0 <
        0"
        Beq $s1, $0, loop

        Jr $ra

#print the integer in $a0 followed by linefeed
Printint:
        Li $v0, 1
        Syscall
        Li $v0, 4 #print_string
        La $a0, lf
        syscall

        Jr $ra

        .data
lf:
        .asciiz "\n"
```

trailed $ra to SC can't get back to SC

STACK

$sp - register points to where in stack



when adding to the stack, always goes down a memory address

```
        .text
main:
        li $s0, 10

Loop:
        move $a0, $s0

        addi $sp, $sp, -4
        sw $ra, $($sp)

        jal printint

        Lw $ra, ($sp)
        addi $sp, $sp, 4

        sub $s0, $s0, 1 # assembler still knows subi
        slt $s1, $s0,$ 0 # set t1 to result  of "t0 <
        0"
        beq $s1, $0, loop

        jr $ra

#print the integer in $a0 followed by linefeed
Printint:
        Li $v0, 1
        Syscall
        Li $v0, 4 #print_string
        La $a0, lf
        syscall

        Jr $ra

        .data
lf:
        .asciiz "\n"
```

push on stack

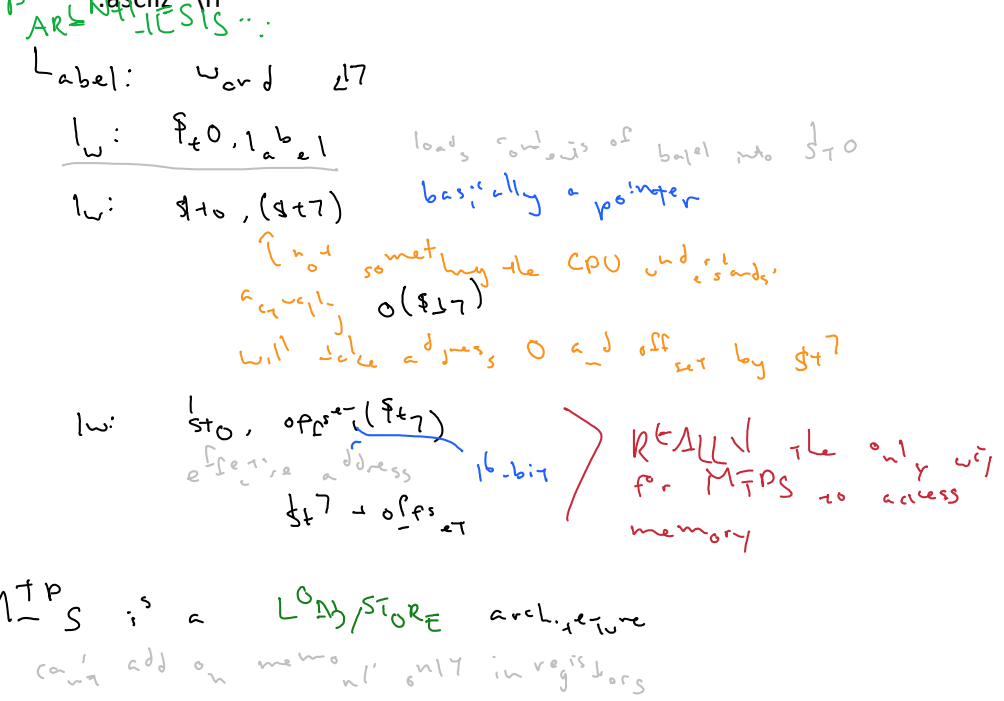
pop from stack

bad placement for stack because we do it in LOOP, we can take it out of the loop

Cd2.s **(a better stack placement)**

```
        .text
main:
        addi $sp, $sp, -4
        sw $ra, $($sp)
        li $s0, 10

Loop:
        move $a0, $s0
        jal printint

        sub $s0, $s0, 1 # assembler still knows subi
        slt $s1, $s0,$ 0 # set t1 to result  of "t0 <
        0"
```

```
        beq $s1, $0, loop

        Lw $ra, ($sp)
        addi $sp, $sp, 4

        jr $ra

#print the integer in $a0 followed by linefeed
Printint:
        Li $v0, 1
        Syscall
        Li $v0, 4 #print_string
        La $a0, lf
        syscall

        Jr $ra

        .data
lf:
        .asciiz "\n"
```



Cd2.s **(saving s registers)**

```
        .text
main:
        addi $sp, $sp, -4
        sw $ra, $($sp)
        addi $sp, $sp, -4
        sw $s0, $($sp)
        li $s0, 10

Loop:
        move $a0, $s0
```

```
        jal printint

        sub $s0, $s0, 1 # assembler still knows subi
        slt $t1, $s0,$ 0 # set t1 to result  of "t0 < 0"
        beq $t1, $0, loop

        Lw $20, ($sp)
        addi $sp, $sp, 4
        Lw $ra, ($sp)
        addi $sp, $sp, 4

        jr $ra

#print the integer in $a0 followed by linefeed
Printint:
        #if I want to use s register, I need to push onto stack,
        otherwise this function is wrong
        Li $v0, 1
        Syscall
        Li $v0, 4 #print_string
        La $a0, lf
        syscall

        Jr $ra

        .data
lf:
        .asciiz "\n"
```

Cd2.s **(instead of one at a time)**

```
        .text
main:
        addi $sp, $sp, -8
        sw $ra, 4($sp)
        sw $s0, 0($sp)

        li $s0, 10

Loop:
        move $a0, $s0
        jal printint

        sub $s0, $s0, 1 # assembler still knows subi
```

```
        slt $t1, $s0,$ 0 # set t1 to result  of "t0 < 0"
        beq $t1, $0, loop

        Lw $s0, 0($sp)
        Lw $ra, 4($sp)
        Addi $sp, $sp, 8

        jr $ra

#print the integer in $a0 followed by linefeed
Printint:
        Li $v0, 1
        Syscall
        Li $v0, 4 #print_string
        La $a0, lf
        syscall

        Jr $ra

        .data
lf:
        .asciiz "\n"
```