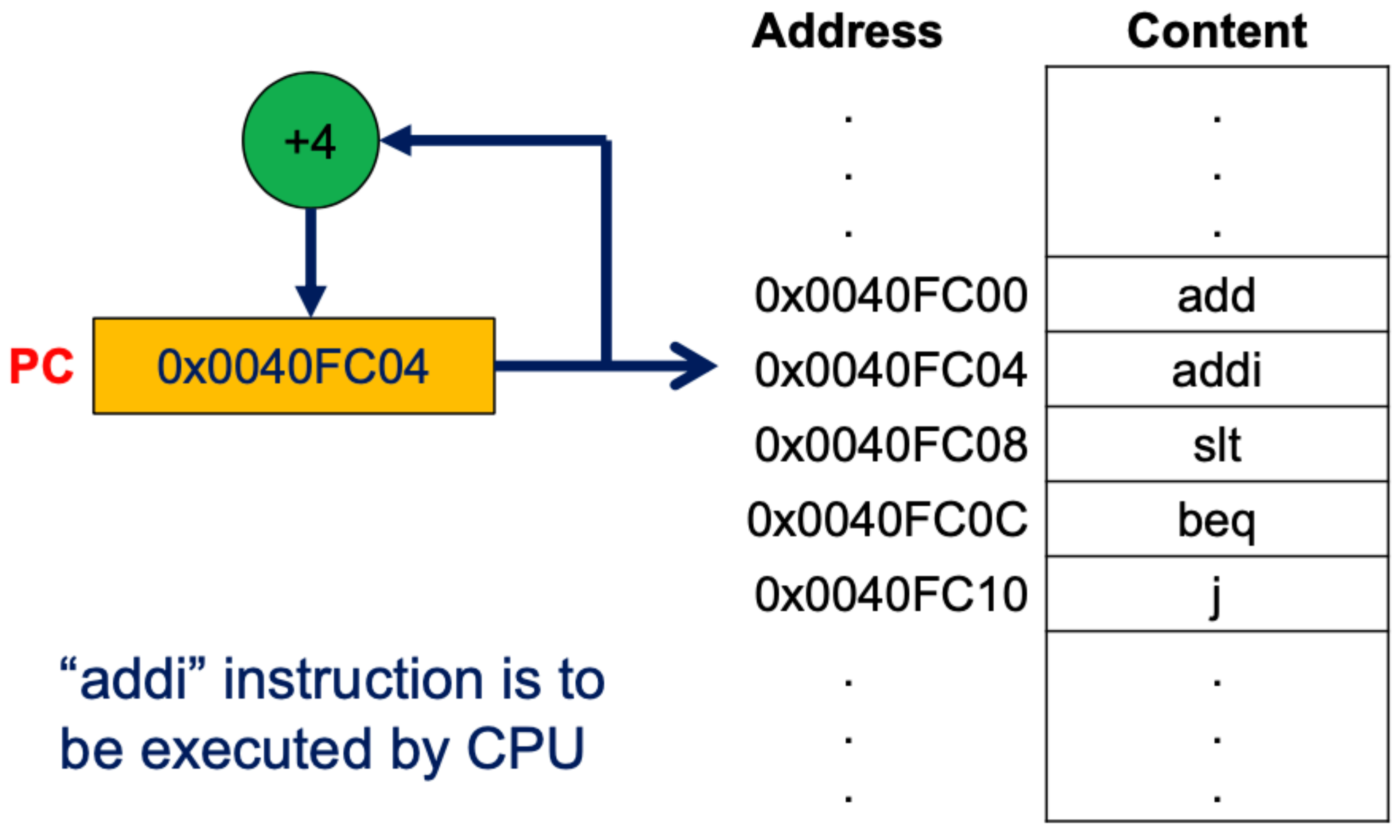
VE370 RC (Week 3)

Li Shi

2020.9.25

Key concept 1: Program counter



“addi” instruction is to be executed by CPU

Program stored in memory

Key concept 2: Function call instructions

- **`jal FunctionLabel`** (J-type instruction)
  - 1. Set **`$ra=PC+4`**
  - 2. Set **`PC=Addr(FunctionLabel)`**
- **`jr $ra`** (R-type instruction)
  - 1. Set **`PC=$ra`**

Key concept 3: Steps of function call

Steps of function call operation

1. Place parameters in parameter registers
2. Transfer control to the function
3. Acquire storage for the function in stack
4. Perform function’s operations
5. Place results in result register(s) for caller
6. Release storage
7. Return to the place before the function call

Key concept 3: Steps of function call

- Review lecture slide 21~24. Do remember all the crucial steps!
- Answer the following questions:
  - What should we do before the function is called?
    - Set **`$a_i`**, save regs, **`jal`**
  - What should we do after function call, but before the function starts executing?
    - Set **`$sp`** (and **`$fp`**), save regs
  - What should we do before the called functions finishes?
    - Set **`$v_i`**, restore regs, set **`$sp`** (and **`$fp`**), **`jr $ra`**

Function call example: `fact`

C code:

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

- Argument n in $a0
- Result in $v0

Some questions for revision:
1. Why is it necessary to save `$ra`?
2. When executing `fact(3)`, what does the stack look like?
3. Could you write a main function to invoke `fact(3)`?

MIPS code:

```
fact:
    addi $sp, $sp, -8     # adjust stack for 2 items
    sw   $ra, 4($sp)      # save return address
    sw   $a0, 0($sp)      # save argument
    slti $t0, $a0, 1      # test for n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1    # if so, result is 1
    addi $sp, $sp, 8      #   release stack
    jr   $ra              #   and return
L1: addi $a0, $a0, -1     # else decrement n
    jal  fact             # recursive call
    lw   $a0, 0($sp)      # restore original n
    lw   $ra, 4($sp)      #   and return address
    addi $sp, $sp, 8      # pop 2 items from stack
    mul  $v0, $a0, $v0    # multiply to get result
    jr   $ra              # and return
```

My experience on MIPS Assembly programming template

- Example 1. Branch structure
  - `if ($s0 < $s1) { ... } else { ... }`
- Example 2. Loop structure
  - `for ($t0 = 0; $t0 < $a1; $t0++) { ... }`
- Example 3. Function call
  - `int fib(int n) { ... return $v0; }`
  - `fib($a0)`

- See blackboard notes. You can make your own reference sheet.

Class exercise: `fib`

- Convert the following C code into MIPS Assembly.

```
int fib(int n) {
  if (n < 3)
    return 1;
  else
    return fib(n-1) + fib(n-2);
}
```

Class exercise: `fib`

Step 1. `int fib(int n) { ... }`

```
fib:
  addi $sp, $sp, -12 # Allocate the stack frame
  sw   $ra, 8($sp)
  sw   $a0, 4($sp)
  sw   $s0, 0($sp)   # We will use $s0 later

  ... (See Step 2)

  lw   $s0, 0($sp)
  lw   $a0, 4($sp)
  lw   $ra, 8($sp)
  addi $sp, $sp, 12  # Pop the stack
  jr   $ra
```

Class exercise: `fib`
Step 2. `if (n < 3) ... else ...`

```
fib:
  addi $sp, $sp, -12 # Allocate the stack frame
  sw   $ra, 8($sp)
  sw   $a0, 4($sp)
  sw   $s0, 0($sp)   # We will use $s0 later
  slti $t0, $a0, 3   # Test for n < 3
  beq  $t0, $0, elseBlock
  ... (See Step 3)

elseBlock:
  ... (See Step 4)

  lw   $s0, 0($sp)
  lw   $a0, 4($sp)
  lw   $ra, 8($sp)
  addi $sp, $sp, 12  # Pop the stack
  jr   $ra
```

Class exercise: `fib`
Step 3. `return 1;`

```
fib:
  addi $sp, $sp, -12 # Allocate the stack frame
  sw   $ra, 8($sp)
  sw   $a0, 4($sp)
  sw   $s0, 0($sp)   # We will use $s0 later
  slti $t0, $a0, 3   # Test for n < 3
  beq  $t0, $0,  elseBlock

  addi $v0, $0,  1   # return 1
  addi $sp, $sp, 12
  jr   $ra
```

Class exercise: `fib`

Step 4. `return fib(n-1)+fib(n-2);`

```
elseBlock:
  addi $a0, $a0, -1
  jal  fib             # fib(n-1)
  addi $s0, $v0, 0     # Q: What is $s0 used for?
  addi $a0, $a0, -1
  jal  fib             # fib(n-2)
  add  $v0, $v0, $s0 # return fib(n-1)+fib(n-2)

  lw   $s0, 0($sp)
  lw   $a0, 4($sp)
  lw   $ra, 8($sp)
  addi $sp, $sp, 12   # Pop the stack
  jr   $ra
```

More challenging class exercise: insertionSort

- Convert the following C code into MIPS Assembly.

```c
void insertionSort(int arr[], int n) {
  int i, key, j;
  for (i = 1; i < n; i++) {
    key = arr[i];
    j = i - 1;
    while (j >= 0 && key < arr[j]) {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
  arr[j + 1] = key;
  }
}
```

Exploratory topic in CS: tail call optimization (Not required in VE370)

- Consider the following C code.
  - We know that when **n** is large, the stack may overflow. What is the approximate value of **n** when the stack overflows?
  - The non-official definition of tail call optimization is given here. How to modify your C code to apply tail call optimization?
  - Apply tail call optimization. Convert your modified C code to MIPS Assembly.

```c
int fib(int n) {
  if (n < 3)
    return 1;
  else
    return fib(n-1) + fib(n-2);
}
```