Computer Systems
#8. Instructions (Control Transfer)

Keiji Kimura <keiji@waseda.jp>

Ans. for Exercise (1)

- Translate following expression into MIPS assembly language, then encode into machine code under the assumption that $t1 has the base of the array A and $s2 corresponds to h:
  - A[300] = h+A[300]; /* All variables are 32bit integer */
    - TBA
- Exploit rt field into $t1 when $s0 has an instruction code.
  - TBA

branch

- Conditional branch
  - Based on the condition at that time, jump to other instructions
- MIPS has two conditional branch instructions
  - beq register1, register2, L1
    - Jump to L1 if register1 equals to register2
  - bne register1, register2, L1
    - Jump to L1 if register 1 doesn't equal to register2

PC relative addressing

- PC (program counter)
  - contains the address of the current instruction.
- beq and bne are I-format

I-format

op	rs	rt	immediate
31 26	25 21	20 16	15 0

  - immediate has only 16bits width
  - Not enough for representing an instruction address (32bits)
- PC relative addressing
  - Target address is calculated as (PC+4)+immediate*4.
    - immediate is signed value.
    - Why (PC+4)? Why immediate*4?
- j (jump instruction)
  - J-format (26bits)
  - "address" represents target address (in word address)

J-format

op	address
31 26	25 0

MIPS Addressing Mode Summary

- Addressing modes
  - multiple forms of addressing
- MIPS addressing modes
  - Immediate addressing
    - The operand is a constant within the instruction itself. (I-format)
  - Register addressing
    - The operand is a register. (R-format)
  - Base or displacement addressing
    - The operand is at the memory location whose address is the sum of a register and a constant in the instruction. (lw/sw)
  - PC-relative addressing
    - The branch address is the sum of the PC and a constant in the instruction.
  - Pseudo direct addressing
    - The jump address is the 26bits of the instruction concatenated with the upper bits of the PC. (J-format)

if-then-else in MIPS assembly

- ex.) if (i==j) f = g+h; else f=g-h;
- Five variables f through j correspond to the five registers $s0 through $s4
-
```
    bne $s3,$s4, Else # if
    add $s0, $s1, $s2 # then part
    j Exit
Else:
     sub $s0, $s1, $s2 # else part
Exit:
```

Loops in MIPS assembly

- Important operation for iterative computations
- ex) while (save[i] == k) i++;
- i and k correspond to register $s3 and $s5 and the base of the array save is in $s6
- Loop:

```
    sll $t1, $s3, 2  # i*4 for save[i] (32bit data)
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit # (save[i] != k)?
    addi $s3, $s3, 1
    j  Loop
Exit:
```

Comparison

- beq and bne are for evaluating equality.
- How about less than and larger than?
- slt: set on less than
  - slt $t0, $s3, $s4 # $t0 = 1 if $s3 < $s4
- slti
  - slti $t0, $s3, 10 # $t0 = 1 if $s3 < 10
- sltu/sltiu
  - for unsigned integers
- How to represent other relative conditions?
  - less than or equal, greater than, greater than or equal
- Why not blt?

Exercise (1)

- Show assembly code for the following loop under the assumption i corresponds to register $s3 and the base of the array “save” is in $s6.
  - for (i = 0; i < 10; i++) save[i] = 0;
- Show assembly code for the following statements under the assumption f through j correspond to the five registers $s0 through $s4.
  - if (i<=j) f = g+h; else f=g-h;
- Suppose register $s0 has the hexadecimal number 0xffffffff and the register $s1 has 1. What are the values of register $t0 and $t1 after the following instructions?
  - slt $t0, $s0, $s1
    sltu $t1, $s0, $s1

Procedure (function) call steps

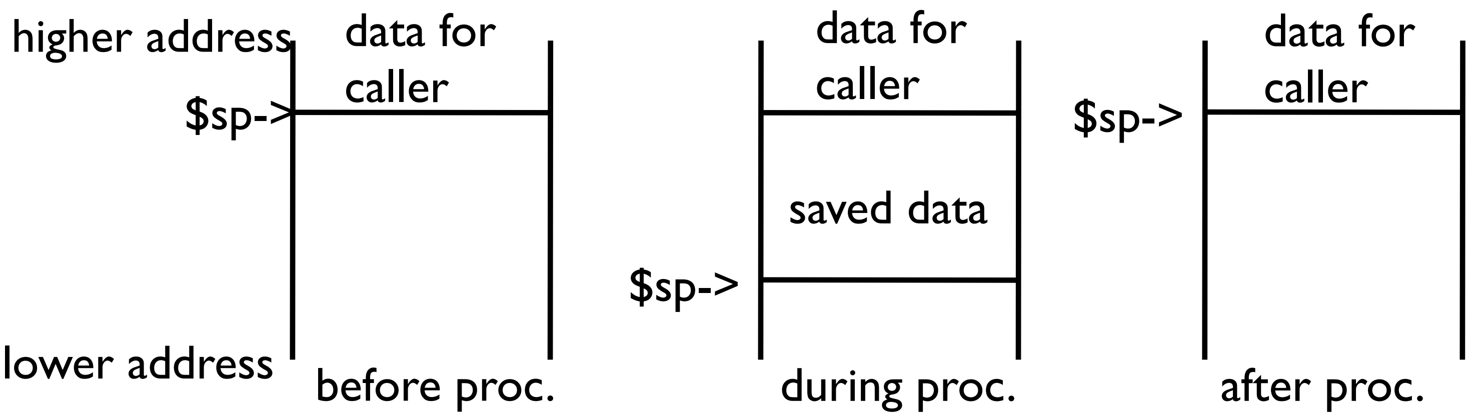
- An important tool programmers use to structure programs
- Go to the procedure, back to the calling point with results, then continue to process after the calling point.
- procedure execution steps:
  - Put parameters in a place where the procedure can access them.
  - Transfer control to the procedure.
  - Acquire the storage resources needed for the procedure.
  - Perform the desired task.
  - Put the result value in a place where the calling program can access it.
  - Return control to the point of origin, since a procedure can be called from several points in a program.

Simple function call and return

- Set arguments to register $a0-$a3.
  - How about arguments more than five?
- jal ProcedureAddress
  - jump-and-link
  - Jumps to an address and simultaneously saves the address of the following instruction (PC+4) in register $ra.
- Set results to register $v0-$v1.
- jr $ra
  - jump register
  - Jumps to the address stored in register $ra.

More registers

- To use other registers than $a0-$a3 and $v0-$v1
  - Save values in registers before procedure processing, and restore them after procedure processing.
  - Stack is used for this purpose.
    - push: save data onto the stack
    - pop: restore (and remove) from the stack
    - stack pointer: the pointer holding the stack top address
      - The register $sp in MIPS



ex) procedure body with push/pop

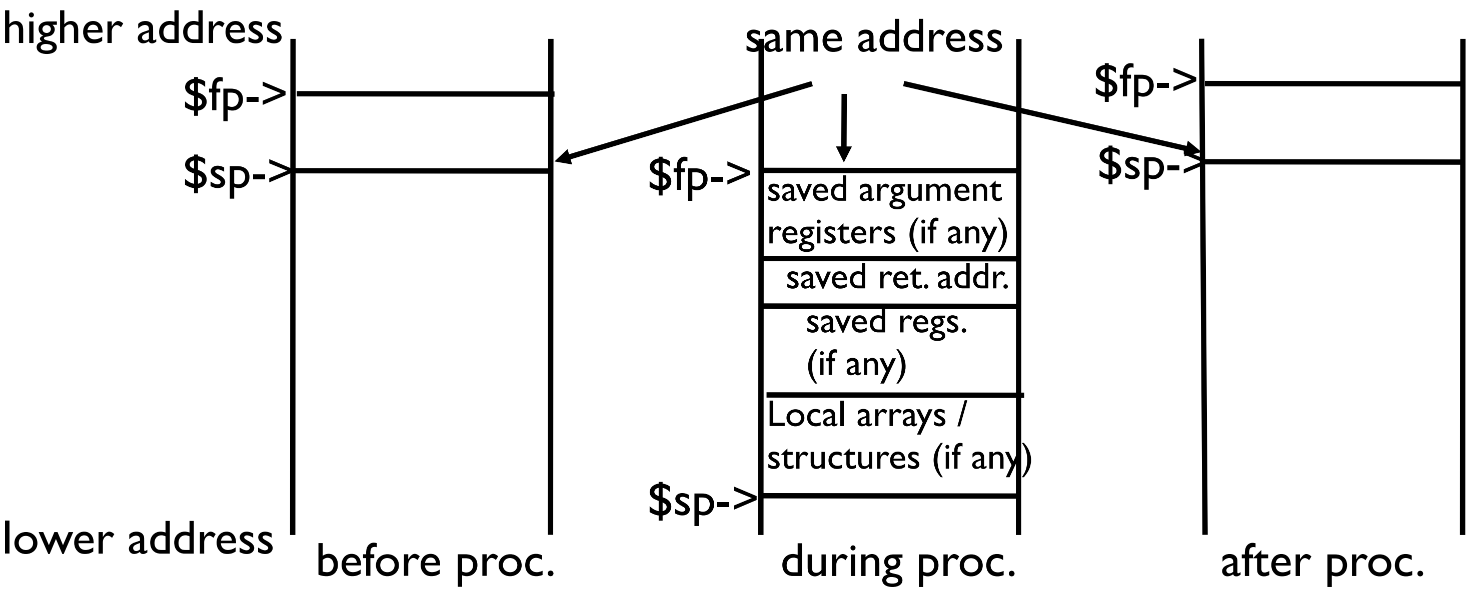
- int leaf_example(int g, int h, int i, int j)
```
{
    int f; /* corresponds $s0 */
    f = (g+h)-(i+j);
    return f;
}
```
- leaf_example:
```
    addi $sp, $sp, -12  # adjust stack to make room for 3 items
    sw  $t1, 8($sp)     # push values
    sw  $t0, 4($sp)
    sw  $s0, 0($sp)
    add $t0, $a0, $a1   # proc. body
    add $t1, $a2, $a3
    sub $s0, $t0, $t1
    add $v0, $s0, $zero  # returns f ($v0 = $s0+0)
    lw  $s0, 0($sp)     # pop vaules
    lw  $t0, 4($sp)
    lw  $t1, 8($sp)
    addi  $sp, $sp, 12   # adjust stack to delete 3 items
    jr  $ra   # back to calling point
```
- Actually, $t0-$t9 need not to save by the callee (called procedure) in MIPS convention.

Nested procedure call

- How about further procedure calls, such as recursive calls?
  - All execution context, including $ra, $sp and other registers, must be saved onto the stack.
- ```
  int fact(int n)
  {
      if (n < 1) return 1;
      else return (n*fact(n-1));
  }
  ```
- ```
  fact:
      addi  $sp, $sp, -8 # adjust stack for 2 items
      sw  $ra, 4($sp)  # save the return address
      sw  $a0, 0($sp)
      slti  $t0, $a0, 1  # test for n < 1
      beq  $t0, $zero, L1
      addi  $v0, $zero, 1  #return 1
      addi  $sp, $sp, 8  # pop 2 items off stack
      jr   $ra
  L1:
      addi  $a0, $a0, -1
      jal  fact
      lw  $a0, 0($sp)  # return from jal
      lw  $ra, 4($sp)
      addi  $sp, $sp, 8
      mul  $v0, $a0, $v0  # return n*fact(n-1)
      jr  $ra
  ```

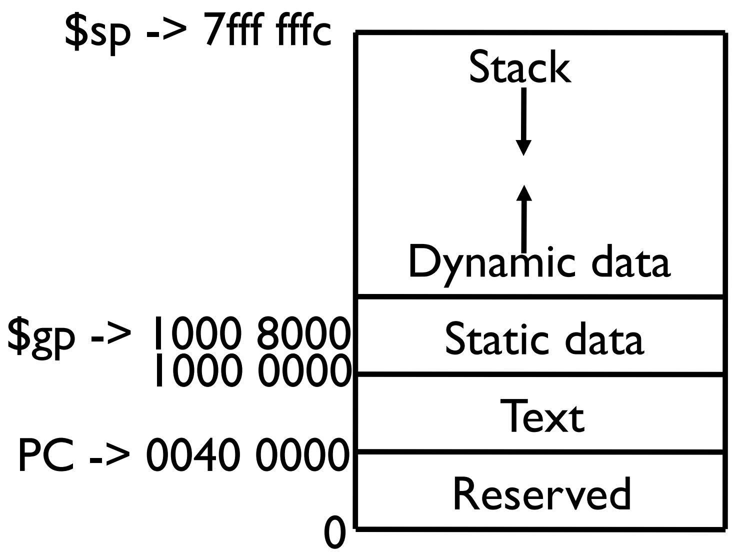
Procedure frame (Activation record)

- Procedure local variables, which do not fit in registers, are also stored in the stack.
- Procedure frame (Activation record)
  - the stack containing a procedure's saved registers and local variables
  - $fp (frame pointer): pointing to the first word of the frame of a procedure.



Global variables and Heap objects

- Global variables and static variables in C
  - the register $gp (global pointer) is used to access these data in MIPS.
- Dynamically allocated objects are from heap area.
  - malloc() in C, new in Java
  - Following is MIPS memory map.



Exercise (2)

- Rewrite assembly code for fact using $fp.
- Then, show the procedure frame before and after the 2nd calling of fact.