CS31001 COMPUTER ORGANIZATION AND ARCHITECTURE

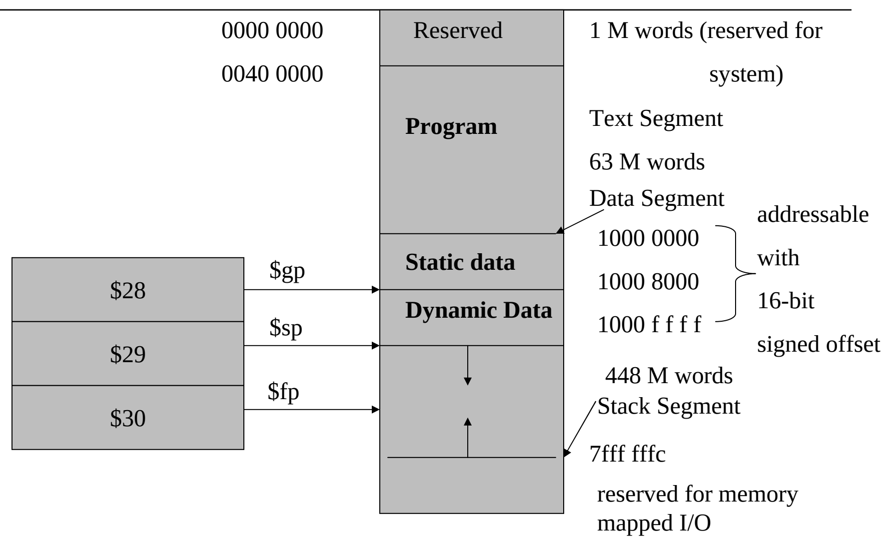
Debdeep Mukhopadhyay,
CSE, IIT Kharagpur

Procedures and Data

The Stack for Data Storage

- We have seen that the stack can be used for procedure calls, and for temporary storage of data.
- Let us see the memory address space in the MIPS memory.

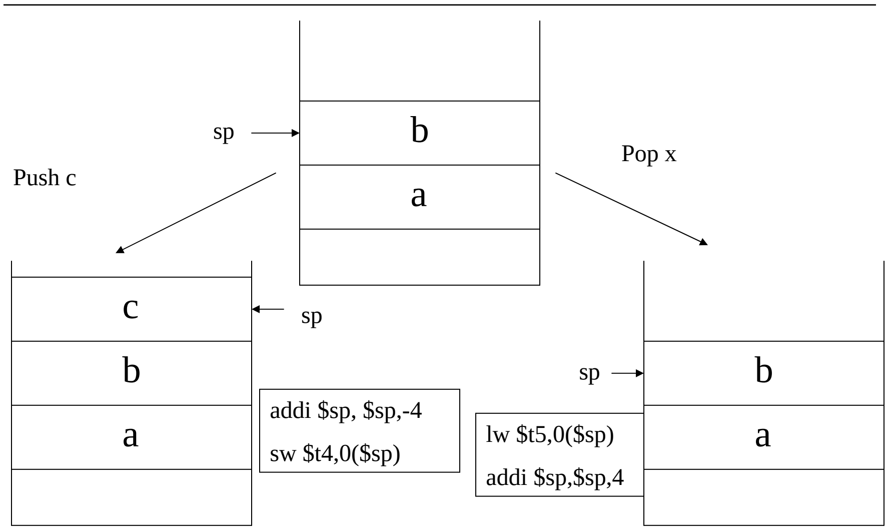
Overview of the memory address space



Push and Pop in Stack

- The stack pointer $sp, points to the top element in the stack.
- Push decrements the stack pointer and puts an element into the stack.
- Pop removes an element from the stack, and then adds the stack pointer.

Effects of push and pop on the stack



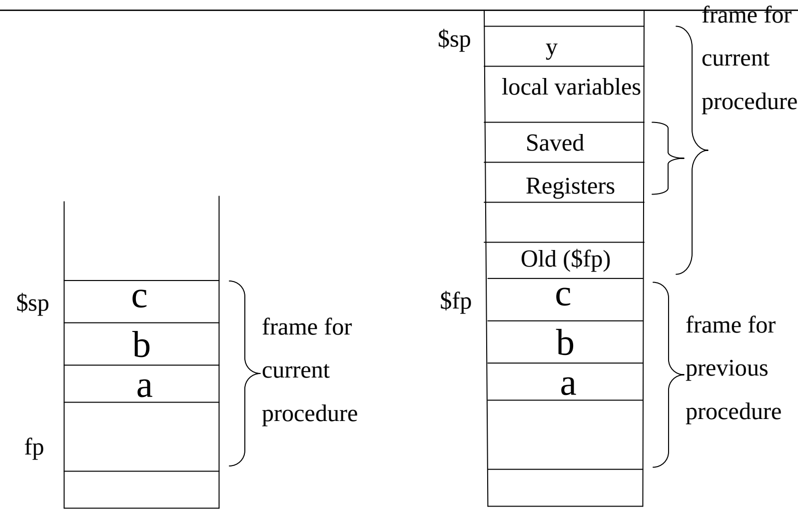
The pop operation

- Note that the pop operation does not erase the old top element, c
- c is still there, and would be still accessible by -4($sp).
- However, it is not a part of the stack frame.
  - Thus this is a logical deletion.
- Hence to delete the top 10 elements, we can (logically) remove them by increasing the stack pointer by 40.

The Stack Frame

- The stack is used for various purposes.
- Two of the important ones are:
  - To pass more than 4 input parameters, or receive more than two results.
  - Place to store when calling other procedures (during nested call).
- Each procedure maintains an area: called as its stack frame.
  - Its delimitors are $sp (top of the stack frame), and $fp(frame-pointer, the other end).
- After the procedure terminates, the calling procedure expects to find the stack undisturbed
  - Thus it can restore the saved registers to their original values and proceed with its own computations.

Use of the stack for procedures



The Frame Pointer

- It provides a stable reference point for addressing memory words in the portion of the stack corresponding to the present procedure.
- The words in the current frame are 4($fp),8($fp),…
- Though the stack pointer changes in the course of the procedure, the frame pointer holds a fixed address.
- **Note that the use of the frame pointer is entirely optional.**

Example

```
proc: sw $fp, -4($sp)
      addi $fp,$sp,0
      addi $sp,$sp,-12
      sw $ra,4($sp)
      sw $s0,0($sp)
      …
      lw $s0,0($sp)
      lw $ra,4($sp)
      addi $sp,$fp,0
      lw $fp,8($sp)
      jr $ra
```

Reduce unnecessary stack operations

```
proc: sw $s0,-4($sp)
      …
      lw $s0, -4($sp)
      jr $ra
```

This reduces the procedure call substantially.

Some Other Instructions

- Special arithmetic/logical instructions
- mult/div
- mult $s0, $s1 #Hi, Lo are set to $s0 x $s1
- div $s0, $s1 #Hi is $s0 mod $s1
- mfhi $t0 #set $t0 to (Hi)
- mflo $t0 #set $t0 to (Lo)

The mult and div instructions

000000	10000	10001	00000	00000	0110x0
ALU Instruction	Source register 1	Source register 2	Unused	Unused	function mult=24, div=26

000000	00000	00000	01000	00000	0100x0
ALU Instruction	Unused	Unused	Destination register	Unused	function mfhi=16 mflo=18

The Shift Operations

000000	00000	10001	01000	00010	0000x0
ALU Instruction	Unused	Source register	Destination Regiister	Shift amount	function sl1=0, sr1=2

000000	10000	10001	01000	00000	0001x0
ALU Instruction	Amount Register	Unused	Destination register	Unused	function sl1v=4 srlv=6

Arrays and Pointers

- Often it is important to step through an array or list.
- Two basic ways:
  - **Index:** Uses a register that holds the index i and increment the register in each step to effect moving from element i of the list to element i+1.
  - **Pointer:** Uses a register that points to (holds the address of) the list element being examined, and updates it each step to point to the next element.

Maximum sum prefix in a list of integers

array base address A in $a0, its length n in $a1.

length of max-sum prefix: $v0

associated sum: $v1

Program

```
mspfx: addi $v0, $zero, 0 #initialize length
       addi $v1, $zero, 0 #initialize max sum
       addi $t0, $zero, 0 #initialize index to 0
       addi $t1, $zero, 0#initialize running sum
loop:  add $t2, $t0, $t0
       add $t2, $t2, $t2
       add $t3, $t2, $a0
       lw $t4, 0($t3)
```

Program

```
add $t1, $t1, $t4
slt $t5, $v1, $t1
bne $t5, $zero, mdfy
j test
mdfy: addi $v0, $v0, 1
        addi $v1, $t1, 0
test: addi $t0, $t0, 1
       slt $t5, $t0,$a1
       bne $t5, $zero, loop
done: jr $ra
```

Selection Sort using Pointers

```
#$a0 pointer to first element in unsorted array
#$a1 pointer to last element in unsorted array
#$t0 temporary place for value of last element
#$v0 pointer to max element in unsorted part
#$v1 value of max element in unsorted part
sort: beq $a0, $a1,done
       jal max
        lw  $t0,0($a1)
        sw $t0,0($v0)
        sw $v1,0($a1)
       addi $a1,$a1,-4
```

Selection Sort using Pointers

```
#$a0 pointer to first element
#$a1 pointer to last element
#$t0 pointer to next element
#$t1 value of next element
#$t2 result of (next) < (max)
#$v0 pointer to max element
#$v1 value of max element
max: addi $v0,$a0,0
     lw $v1,0($v0)
     addi $t0,$a0,0
loop: beq $t0,$a1,ret
      addi $t0,$t0,4
      lw $t1,0($t0)
      slt $t2,$t1,$v1
      bne $t2,$zero,loop
      addi $v0,$t0,0 #update
      addi $v1,$t1,0 #new max
      j loop
ret: jr $ra
```

Rest of Procedures, refer to Tutorial provided in the Lab.