

## Class 7: Intro to SRC Simulator Procedure Calls HLL -> Assembly

## Announcements

- Homework #2
  - Due next Wednesday, Sept. 13 in class
- Class notes posted on course website

## Agenda

- SRC and Procedure Calls
  - Calling convention discussed here will be used for SRC programming lab
- Sample problems from Chapter 2

## SRC – Procedure Calls

- Activation stack
  - Memory layout
  - Activation records
- Link register
  - r31 for this example
- Calling conventions
  - Caller-save (other registers)
  - Callee-save (SP for this example)

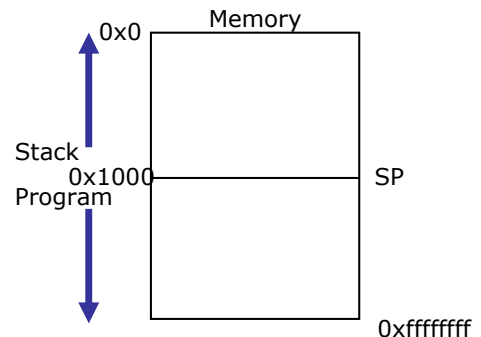
## High Level Program

```
int arg1 = -5; // argument to pass
int output = 0; // result

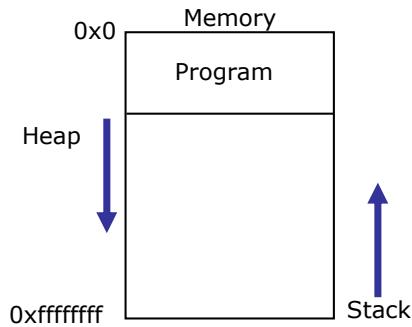
int absolute(int param1) {
    if(param1 <= 0)
        param1 = -param1;
}

void main() {
    output = absolute(arg1);
}
```

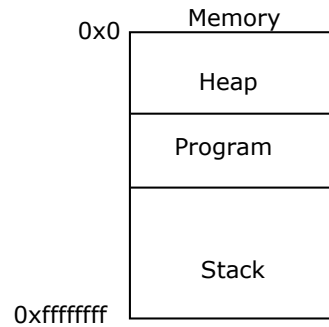
## SRC Memory Layout



## Typical Memory Layout



## Why Wouldn't This Layout Be As Good?



## Activation Stack

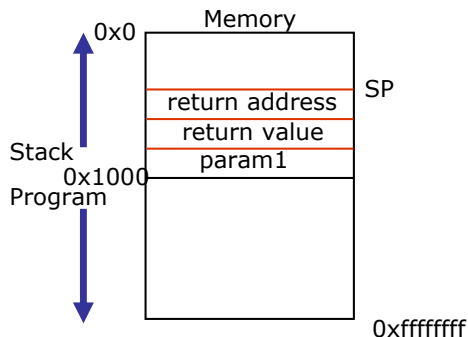
- Stack
  - Last In First Out (LIFO)
    - Push, pop operations
  - Can be located anywhere in memory
    - Typically started in high memory
    - For SRC, we'll use a different memory layout

## What is Saved on the Stack?

- Activation Records
  - Return address
  - Function parameters
  - Local variables
  - Space for return values

Information associated with a procedure call

## Activation Record



## Caller vs. Callee

- Caller
  - Calling function
- Callee
  - Function being called

## High Level Program

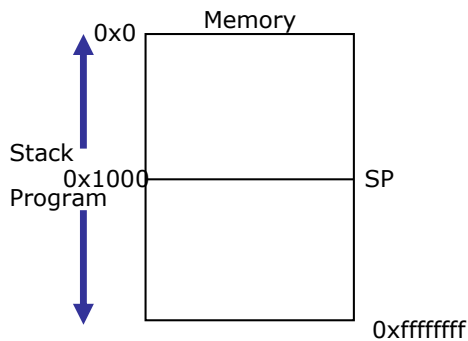
```
int arg1 = -5; // argument to pass
int output = 0; // result
```

```
int absolute(int param1){
    if(param1 <= 0)
        param1 = -param1;
    return param1;
}
void main(){
    output = absolute(arg1);
}
```

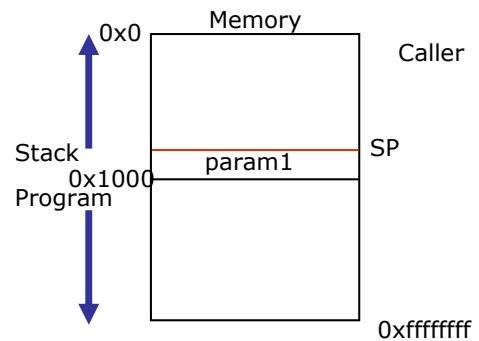
## How is a Procedure Call Performed?

- Caller
  1. Stores arguments onto stack
  2. Allocates space for output values (return value)
  3. Calls the procedure
- Callee
  1. Stores SP
  2. Stores the return address
  3. Retrieves any passed arguments
  4. Performs procedure body
  5. Saves results
  6. Returns to caller

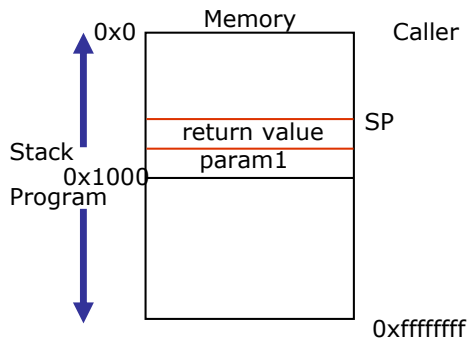
## procedure\_call.asm



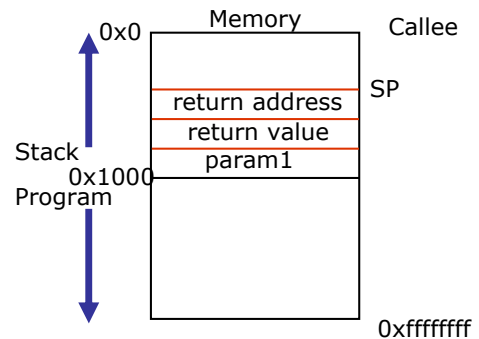
## procedure\_call.asm

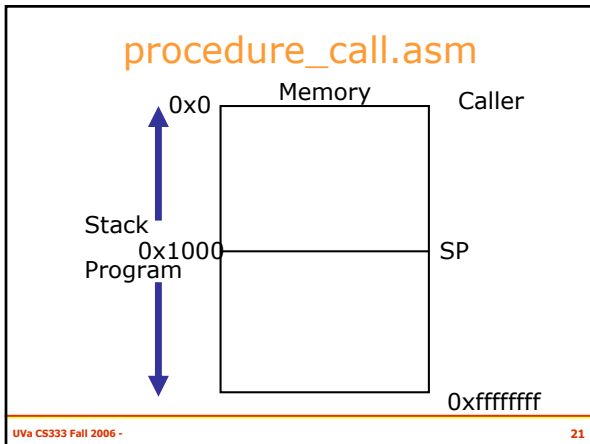
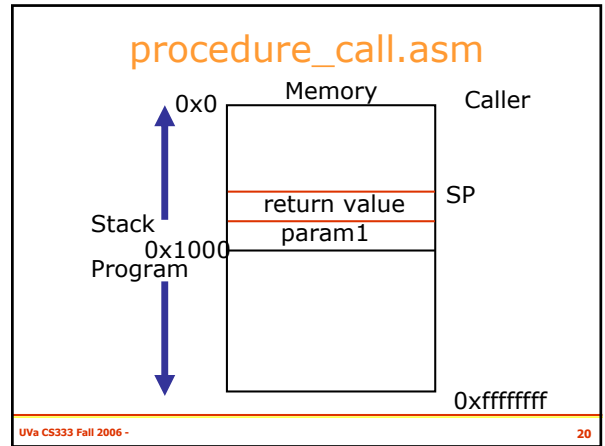
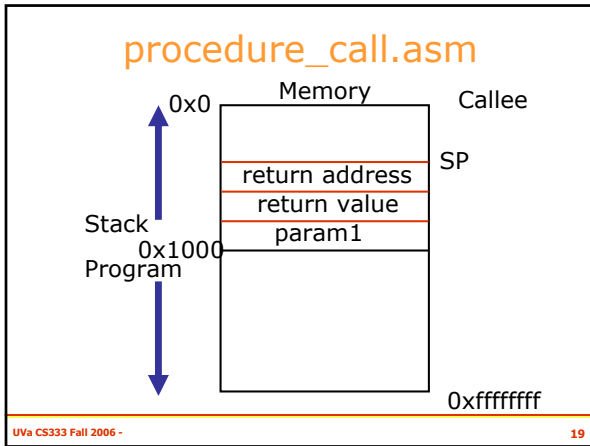


## procedure\_call.asm



## procedure\_call.asm





### Assembly Program (1)

```

arg1:      .dc -5      ; global var arg1 = -5
output:    .dc 0       ; global var output = 0

.org 0x1000 ; start next inst at 0x1000
addi r30, r30, 0x1000; set SP to 0x1000
Main: ld r0, arg1    ; load input, arg1 into r0
addi r30, r30, -4    ; PUSH: adjust SP down by 4 bytes
st r0, (r30)         ; PUSH: save arg1 onto the stack
addi r30, r30, -4    ; allocate space for the
                    ; output (one result)
la r29, Absolute    ; load the address of Abs into r29
brl r31, r29        ; call Absolute and save return
                    ; address (PC+4) in r31

```

UVa CS333 Fall 2006 - 22

### Assembly Program (2)

```

ld r0, (r30)        ; load output result
st r0, output       ; store output result
addi r30, r30, 8    ; restore the SP
stop                ; end of Main program

Absolute:
addi r30, r30, -4   ; PUSH: adjust SP
st r31, (r30)      ; PUSH: save ret addr
                    ; to the stack
la r28, Ret         ; load address of Ret into r28
ld r0, 8(r30)       ; load arg1 from SP+8 into
                    ; param1 space, r0
brpl r28, r0        ; branch to Ret if arg1 >= 0

```

UVa CS333 Fall 2006 - 23

### Assembly Program (3)

```

Negate:  neg r0, r0  ; negate param1
                    ; (get the absolute value)

Ret:     st r0, 4(r30) ; store result r0 (param1)
                    ; onto stack
ld r29, (r30)         ; POP: put the ret addr
                    ; into r29
addi r30, r30, 4     ; POP: adjust SP
br r29                ; return to calling program

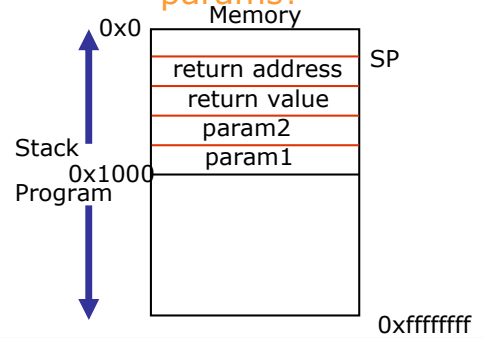
```

UVa CS333 Fall 2006 - 24

## SRC Simulator Demo

- procedure\_call.asm

## Question: What if there were 2 params?



## What Needs to be Agreed Upon?

- Caller and Callee need to agree on
  - Size of inputs and outputs
  - Relative position on the stack (of inputs/outputs)
  - Which registers need to be saved by WHOM
    - **Caller** needs to save any registers that **callee** might use
  - Whether arguments and/or return values are passed through memory or through registers

Caller needs to know how to lay out the stack so that the callee knows how to access its parameters

## What You Should Know

- Activation stack
  - Memory layout
  - Activation records
- Link register
  - r31 for this example
- Calling conventions
  - What actions caller and callee perform
  - Caller-save (other registers)
  - Callee-save (SP in this example)
- Process for converting a HLL program to assembly program

## Chapter 2

### Sample Problems

## Calculate Total Memory Traffic

- Assumptions:
  - Addresses: 2 bytes
  - Data: 2 bytes
  - Opcodes: 1 byte
- Reference
  - Figs 2.4-2.7
- Calculation Steps
  - Instruction Fetch
  - Instruction Execute
  - Total

$$A = (B * C) + D - E$$

3-Address	2-Address	Accumulator	Stack
add a, b, c	load a, b	lda b	push b
mpy a, a, d	add a, c	add c	push c
sub a, a, e	mpy a, d	mpy d	add
	sub a, e	sub e	push d
		sta a	mpy
			push e
			sub
			pop a

## Problem 2.22a

- Modify SRC to include conditional jump instruction, `jpr` (`op=25`)
  - Uses relative addressing, `rel`, instead of branch target register
  - Should only be taken if `ra = 0`
  - See text for full problem desc
  - See Table 2.7 for notation meanings
    - Example of RTN for a different inst:  
`add (:= op = 12) → R[ra] ← R[rb] + R[rc]:`

## Solution

`jpr (:= op=25) → ((ra=0) → PC ← rel) :`