Lecture Link (Video)
https://web.microsoftstream.com/video/6238d56e-29c4-431e-a04b-ada7407761c0

CS222: Computer Architecture

Instructors:
Dr Ahmed Shalaby http://bu.edu.eg/staff/ahmedshalaby14#

الاحترام - الادب - الاخلاق
الطالب - المعيد - الدكتور

Function Calls

- **Caller:** calling function (in this case, `main`)
- **Callee:** called function (in this case, `sum`)

C Code

```
void main()
{
  int y;
  y = sum(42, 7);
  ...
}

int sum(int a, int b)
{
  return (a + b);
}
```

Function Conventions

- **Caller:**
  - passes **arguments** to callee
  - jumps to callee
- **Callee:**
  - **performs** the function
  - **returns** result to caller
  - **returns** to point of call
  - **must not overwrite** registers or memory needed by caller

MIPS Function Conventions

- **Call Function:** jump and link (`jal`)
- **Return** from function: jump register (`jr`)
- **Arguments**: `$a0 – $a3`
- **Return value**: `$v0`

Name	Register Number	Usage
`$0`	0	the constant value 0
`$at`	1	assembler temporary
`$v0-$v1`	2-3	Function return values
`$a0-$a3`	4-7	Function arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved variables
`$t8-$t9`	24-25	more temporaries
`$k0-$k1`	26-27	OS temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	Function return address

Function Calls

C Code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204       add  $s0, $s1, $s2
...

0x00401020 simple: jr $ra
```

`void` means that `simple` doesn't return a value

`jal`: jumps to `simple`
`$ra` = PC + 4 = 0x00400204

Name	Register Number	Usage
`$ra`	31	Function return address

`jr $ra`: jumps to address in `$ra` (0x00400204)

Input Arguments & Return Value

MIPS conventions:

- Argument values: `$a0` - `$a3`
- Return value: `$v0`

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	Function return values
$a0-$a3	4-7	Function arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	Function return address

Input Arguments & Return Value

C Code

```
int main()
{
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 arguments
  ...
}

int diffofsums(int f, int g, int h, int i)
{
  int result;
  result = (f + g) - (h + i);
  return result;                // return value
}
```

Input Arguments & Return Value

MIPS assembly code

```
# $s0 = y

main:
  ...
  addi $a0, $0, 2     # argument 0 = 2
  addi $a1, $0, 3     # argument 1 = 3
  addi $a2, $0, 4     # argument 2 = 4
  addi $a3, $0, 5     # argument 3 = 5
  jal  diffofsums     # call Function
  add  $s0, $v0, $0   # y = returned value
  ...

# $s0 = result
diffofsums:
  add $t0, $a0, $a1   # $t0 = f + g
  add $t1, $a2, $a3   # $t1 = h + i
  sub $s0, $t0, $t1   # result = (f + g) - (h + i)
  add $v0, $s0, $0    # put return value in $v0
  jr  $ra             # return to caller
```

```
int main()
{
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 arguments
  ...
}
```

```
int diffofsums(int f, int g, int h, int i)
{
  int result;
  result = (f + g) - (h + i);
  return result;                // return value
}
```

Input Arguments & Return Value

MIPS assembly code

```
# $s0 = result
diffofsums:
  add $t0, $a0, $a1   # $t0 = f + g
  add $t1, $a2, $a3   # $t1 = h + i
  sub $s0, $t0, $t1   # result = (f + g) - (h + i)
  add $v0, $s0, $0    # put return value in $v0
  jr  $ra             # return to caller
```

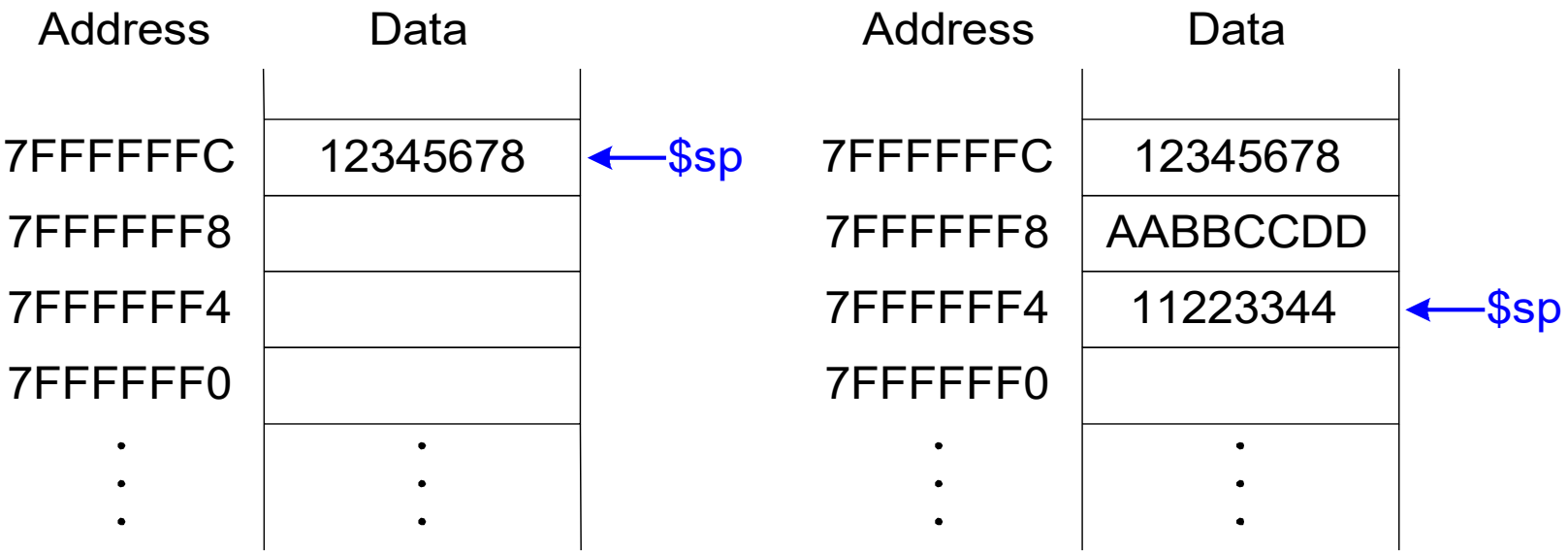
- `diffofsums` overwrote 3 registers: `$t0`, `$t1`, `$s0`
- `diffofsums` can use *stack* to temporarily store registers

The Stack

- Memory used to temporarily save variables
- Like stack of dishes, last-in-first-out (LIFO) queue
- ***Expands*:** uses more memory when more space needed
- ***Contracts*:** uses less memory when the space is no longer needed

The Stack

- Grows down (from higher to lower memory addresses)
- Stack pointer: `$sp` points to top of the stack



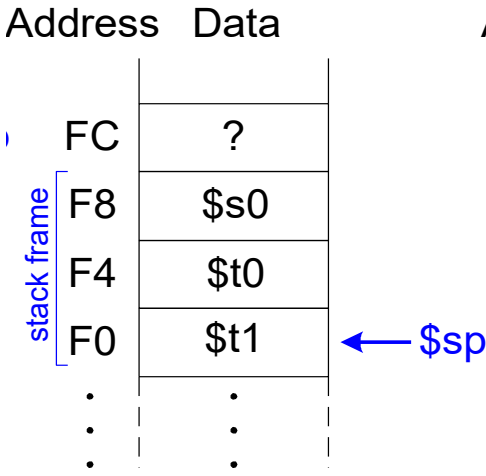
How Functions use the Stack

- Called functions must have no unintended side effects
- But `diffofsums` overwrites 3 registers: `$t0`, `$t1`, `$s0`

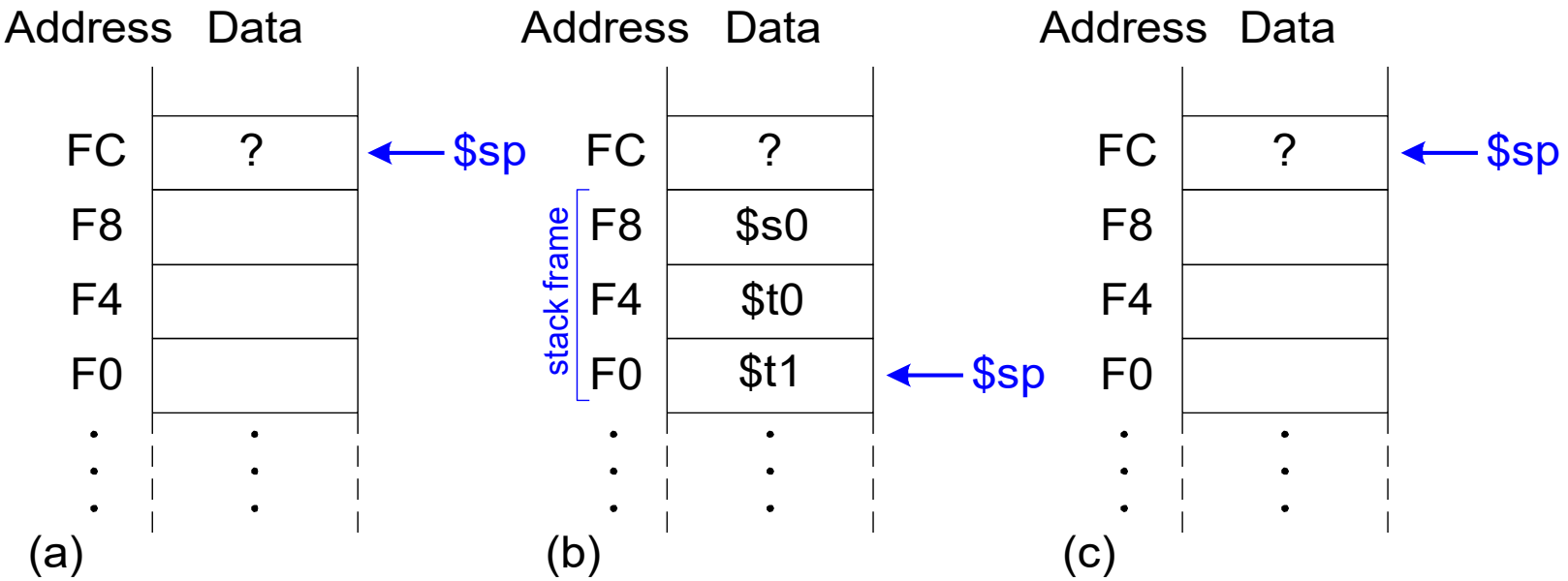
```
# MIPS assembly
# $s0 = result
diffofsums:
  add $t0, $a0, $a1   # $t0 = f + g
  add $t1, $a2, $a3   # $t1 = h + i
  sub $s0, $t0, $t1   # result = (f + g) - (h + i)
  add $v0, $s0, $0    # put return value in $v0
  jr  $ra             # return to caller
```

Storing Register Values on the Stack

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -12   # make space on stack
                       # to store 3 registers
  sw   $s0, 8($sp)     # save $s0 on stack
  sw   $t0, 4($sp)     # save $t0 on stack
  sw   $t1, 0($sp)     # save $t1 on stack
  add  $t0, $a0, $a1   # $t0 = f + g
  add  $t1, $a2, $a3   # $t1 = h + i
  sub  $s0, $t0, $t1   # result = (f + g) - (h + i)
  add  $v0, $s0, $0    # put return value in $v0
  lw   $t1, 0($sp)     # restore $t1 from stack
  lw   $t0, 4($sp)     # restore $t0 from stack
  lw   $s0, 8($sp)     # restore $s0 from stack
  addi $sp, $sp, 12    # deallocate stack space
  jr   $ra             # return to caller
```



The stack during `diffofsums` Call



Registers

Preserved *Callee-Saved*	Nonpreserved *Caller-Saved*
`$s0-$s7`	`$t0-$t9`
`$ra`	`$a0-$a3`
`$sp`	`$v0-$v1`
stack above `$sp`	stack below `$sp`

Function Call Summary

- **Caller**
  - Put arguments in `$a0-$a3`
  - Save any needed registers (`$ra`, maybe `$t0-t9`)
  - `jal callee`
  - Restore registers
  - Look for result in `$v0`
- **Callee**
  - Save registers that might be disturbed (`$s0-$s7`)
  - Perform function
  - Put result in `$v0`
  - Restore registers
  - `jr $ra`

Addressing Modes

How do we address the operands?

- Register Only
- Immediate
- Base Addressing
- PC-Relative
- Pseudo Direct

Addressing Modes

Register Only

- Operands found in registers
  - **Example:** `add $s0, $t2, $t3`
  - **Example:** `sub $t8, $s1, $0`

Immediate

- 16-bit immediate used as an operand
  - **Example:** `addi $s4, $t5, -73`
  - **Example:** `ori  $t3, $t7, 0xFF`

Addressing Modes

Base Addressing

- Address of operand is:

  `base address + sign-extended immediate`

  - **Example:** `lw $s4, 72($0)`
    - address = `$0 + 72`
  - **Example:** `sw $t2, -25($t1)`
    - address = `$t1 - 25`

Addressing Modes

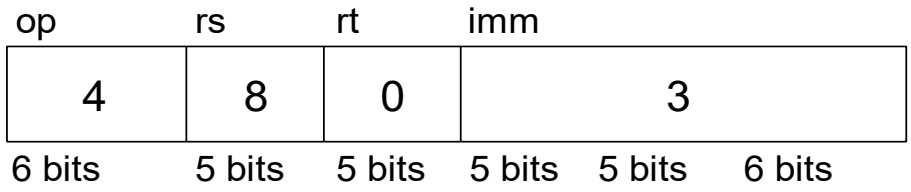
PC-Relative Addressing

```
0x10                beq   $t0, $0, else
0x14                addi  $v0, $0, 1
0x18                addi  $sp, $sp, i
0x1C                jr          $ra
0x20        else:   addi  $a0, $a0, -1
0x24                jal   factorial
```

Assembly Code

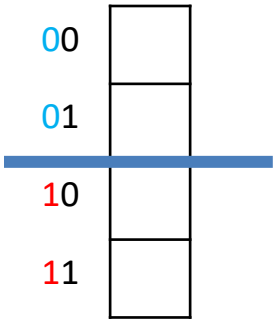
```
beq $t0, $0, else
(beq $t0, $0, 3)
```

Field Values



Addressing Modes

Pseudo-direct Addressing




```
0x0040005C          jal   sum
...
0x004000A0  sum:    add   $v0, $a0, $a1
```

Jump Target Address JTA 0000 0000 0100 0000 0000 0000 1010 0000 (0x004000A0)

26-bit addr 0000 0000 0100 0000 0000 0000 1010 0000 (0x0100028)

0 1 0 0 0 2 8

0	1	2	3

Field Values

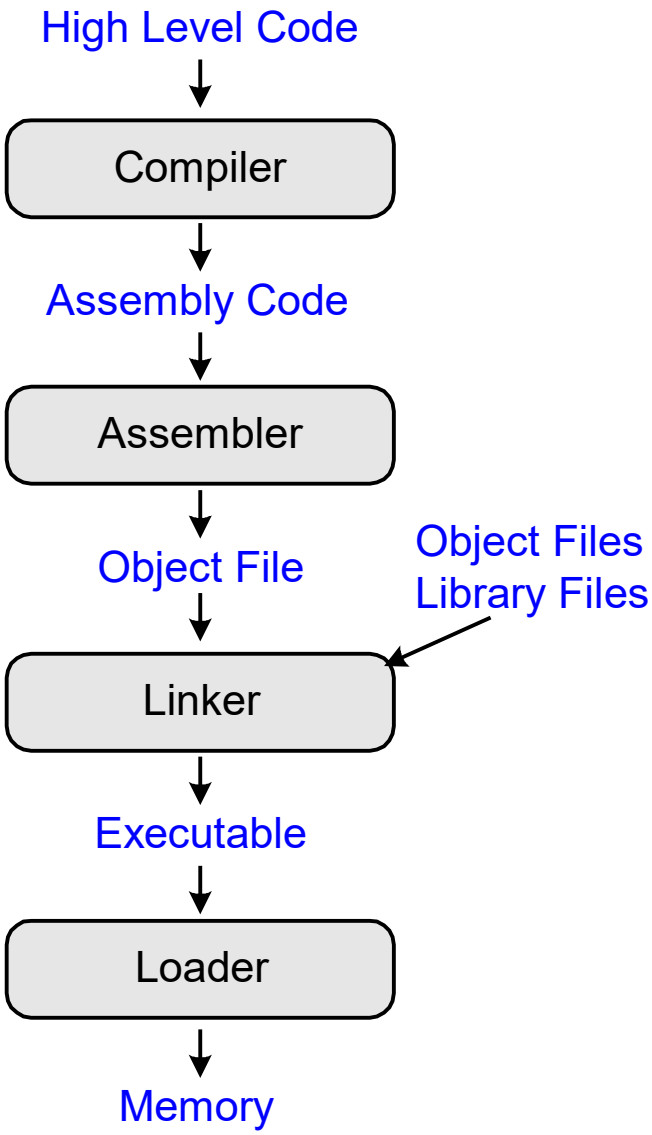
op	imm
3	0x0100028
6 bits	26 bits

Machine Code

op	addr	
000011	00 0001 0000 0000 0000 0010 1000	(0x0C100028)
6 bits	26 bits	

000011 (3)	jal label	jump and link	$ra = PC + 4, PC = JTA

How to Compile & Run a Program

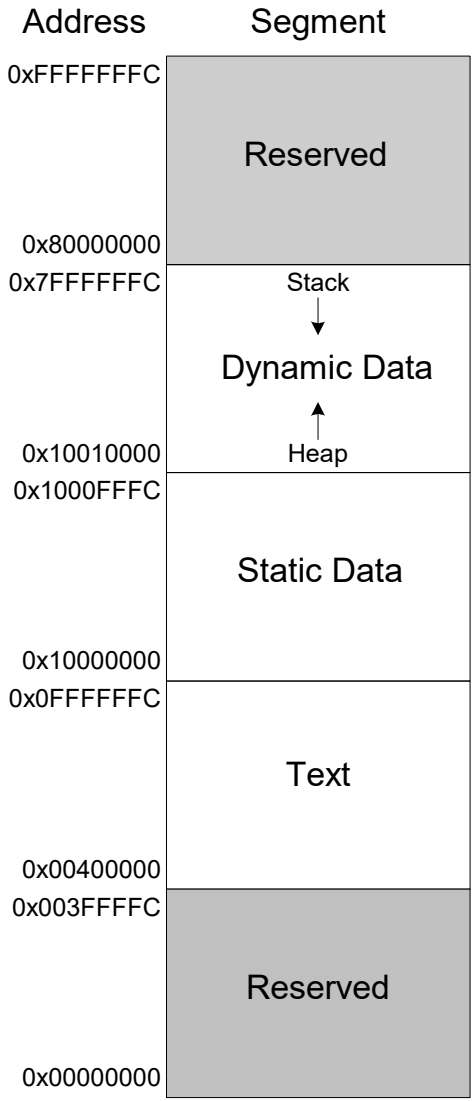


What is Stored in Memory?

- Instructions (also called *text*)
- Data
  - Global/static: allocated before program begins
  - Dynamic: allocated within program

- How big is memory?
  - At most 2^{32} = 4 gigabytes (4 GB)
  - From address 0x00000000 to 0xFFFFFFFF

MIPS Memory Map



Example Program: C Code

```
int f, g, y;  // global variables


int main(void)
{
  f = 2;
  g = 3;
  y = sum(f, g);

  return y;
}


int sum(int a, int b) {
  return (a + b);
}
```

Example Program: MIPS Assembly

```
int f, g, y;  // global


int main(void)
{


  f = 2;
  g = 3;

  y = sum(f, g);
  return y;
}

int sum(int a, int b) {
  return (a + b);
}
```

```
.data
f:
g:
y:
.text
main:
  addi $sp, $sp, -4    # stack frame
  sw   $ra, 0($sp)     # store $ra
  addi $a0, $0, 2      # $a0 = 2
  sw   $a0, f          # f = 2
  addi $a1, $0, 3      # $a1 = 3
  sw   $a1, g          # g = 3
  jal  sum             # call sum
  sw   $v0, y          # y = sum()
  lw   $ra, 0($sp)     # restore $ra
  addi $sp, $sp, 4     # restore $sp
  jr   $ra             # return to OS
sum:
  add  $v0, $a0, $a1   # $v0 = a + b
  jr   $ra             # return
```

Example Program: Symbol Table

Symbol	Address
`f`	**0x10000000**
`g`	**0x10000004**
`y`	**0x10000008**
`main`	**0x00400000**
`sum`	**0x0040002C**

Example Program: Executable

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)

Text segment	Address	Instruction
	0x00400000	0x23BDFFFC
	0x00400004	0xAFBF0000
	0x00400008	0x20040002
	0x0040000C	0xAF848000
	0x00400010	0x20050003
	0x00400014	0xAF858004
	0x00400018	0x0C10000B
	0x0040001C	0xAF828008
	0x00400020	0x8FBF0000
	0x00400024	0x23BD0004
	0x00400028	0x03E00008
	0x0040002C	0x00851020
	0x00400030	0x03E00008

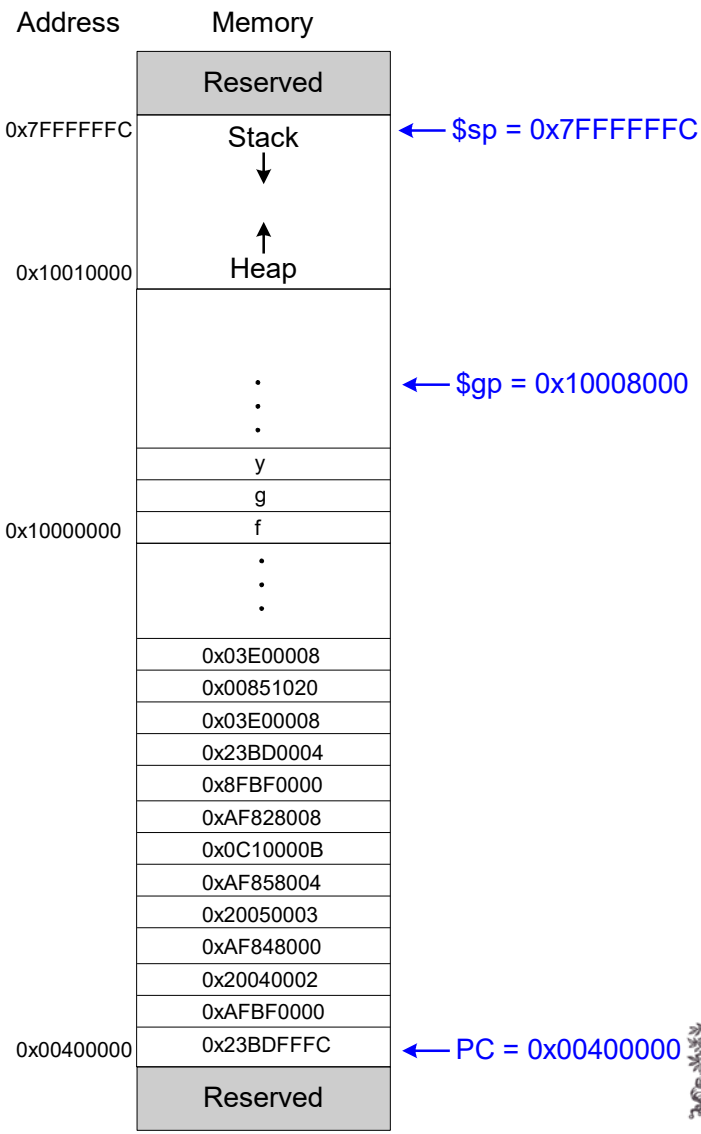
Data segment	Address	Data
	0x10000000	f
	0x10000004	g
	0x10000008	y

```
addi $sp, $sp, -4                    1
sw   $ra, 0 ($sp)                    2
addi $a0, $0, 2                      3
sw   $a0, 0x8000 ($gp)               4
addi $a1, $0, 3                      5
sw   $a1, 0x8004 ($gp)               6
jal  0x0040002C                      7
sw   $v0, 0x8008 ($gp)               8
lw   $ra, 0 ($sp)                    9
addi $sp, $sp, -4                    10
jr   $ra                             11
add  $v0, $a0, $a1                   12
jr   $ra                             13
```

Example Program: In Memory

Instruction	
0x23BDFFFC	addi $sp, $sp, -4
0xAFBF0000	sw $ra, 0 ($sp)
0x20040002	addi $a0, $0, 2
0xAF848000	sw $a0, 0x8000 ($gp)
0x20050003	addi $a1, $0, 3
0xAF858004	sw $a1, 0x8004 ($gp)
0x0C10000B	jal 0x0040002C
0xAF828008	sw $v0, 0x8008 ($gp)
0x8FBF0000	lw $ra, 0 ($sp)
0x23BD0004	addi $sp, $sp, -4
0x03E00008	jr $ra
0x00851020	add $v0, $a0, $a1
0x03E00008	jr $ra

Data
f
g
y



Looking Ahead

Microarchitecture – building MIPS processor in hardware

Bring colored pencils