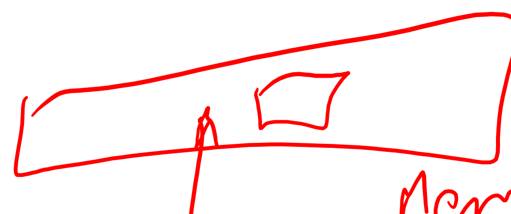
Lecture 5: More Instructions, Procedure Calls

- Today's topics:
  - Numbers, control instructions
  - Procedure calls

lw $t0, 4($gp)
sw
add
addi
sub

HW 2 posted
Due in a week
Using Gradescope for submissions



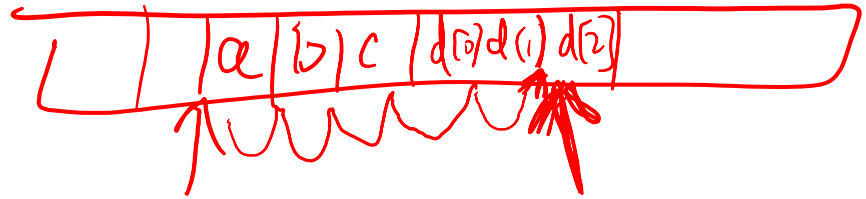
Mem

Example

int a, b, c, d[10]

4 Bytes

Convert to assembly:



$gp

base address

C code: d[3] = d[2] + a;

Assembly (same assumptions as previous example):

```
lw     $s0, 0($gp)      #  a is brought into $s0
lw     $s1, 20($gp)     #  d[2] is brought into $s1
add    $s2, $s0, $s1    #  the sum is in $s2
sw     $s2, 24($gp)     #  $s2 is stored into d[3]
```

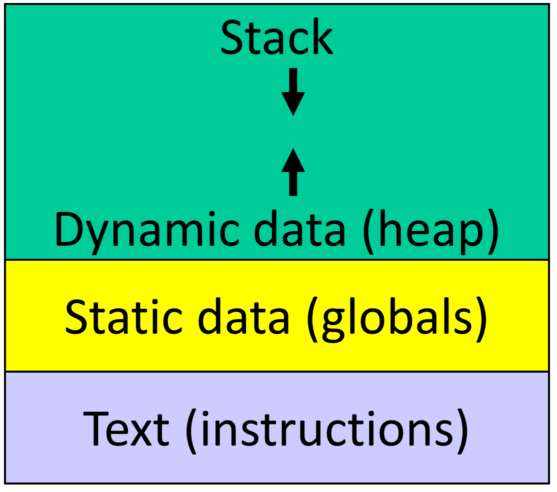
dest is in mem

src src

Assembly version of the code continues to expand!

Memory Organization

- The space allocated on stack by a procedure is termed the activation record (includes saved values and data local to the procedure) – frame pointer points to the start of the record and stack pointer points to the end – variable addresses are specified relative to $fp as $sp may change during the execution of the procedure
- $gp points to area in memory that saves global variables
- Dynamically allocated storage (with malloc()) is placed on the heap



Recap – Numeric Representations

0x 45

0100 0101

0x 7f

0111 1111

- Decimal $35_{10} = 3 \times 10^1 + 5 \times 10^0$
- Binary $00100011_2 = 1 \times 2^5 + 1 \times 2^1 + 1 \times 2^0$
- Hexadecimal (compact representation)

code for hex $0x\,23$ or $23_{hex} = 2 \times 16^1 + 3 \times 16^0$

0-15 (decimal) → 0-9, a-f (hex)

Dec	Binary	Hex	Dec	Binary	Hex	Dec	Binary	Hex	Dec	Binary	Hex
0	0000	00	4	0100	04	8	1000	08	12	1100	0c
1	0001	01	5	0101	05	9	1001	09	13	1101	0d
2	0010	02	6	0110	06	10	1010	0a	14	1110	0e
3	0011	03	7	0111	07	11	1011	0b	15	1111	0f

Instruction Formats

R
I
J

$-32K \longrightarrow +32K$

4B

Instructions are represented as 32-bit numbers (one word), broken into 6 fields

dest

32 regs
5 bits
to rep
each reg
$2^5 = 32$

R-type instruction add $t0, $s1, $s2

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
op	rs	rt	rd	shamt	funct
opcode	source	source	dest	shift amt	function

variant

reg 0 00000
00001

I-type instruction lw $t0, 32($s3)

addi $t1, $t2, 4000

6 bits	5 bits	5 bits	16 bits
opcode	rs	rt	constant
	($s3)	($t0)	

2^{16} → 64K

reg 31 11111

Logical Operations

7

70

sh-left ⇒ mult by 10

Logical ops	C operators	Java operators	MIPS instr
Shift Left	<<	<<	sll
Shift Right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor (with $zero)

→ imm

$t0 5 0......000101

$\boxed{20}$ sll $t1, $t0, 2 0...00 10100 $t1

$\boxed{2}$ srl $t2, $t0, 1 0,-........0010 $t2

$5 \div 2 = 2$

Control Instructions

- Conditional branch: Jump to instruction L1 if register1 equals register2: beq register1, register2, L1
  Similarly, bne and slt (set-on-less-than)

- Unconditional branch:
  j L1
  jr $s0 (useful for big jumps and procedure returns)

Convert to assembly:

```
if (i == j)
   f = g+h;
else
   f = g-h;
```

Control Instructions

PC 4 reg 8 12

32 regs

Mem

- Conditional branch: Jump to instruction L1 if register1 equals register2:   beq   register1, register2, L1
  Similarly, bne and slt (set-on-less-than)

- Unconditional branch:
  j   L1
  jr $s0   (useful for big jumps and procedure returns)

Convert to assembly:

```
if  (i == j)            PC = 48       bne   $s3, $s4, Else    8
    f = g+h;            52 Then:      add   $s0, $s1, $s2     4
else                    56            j     End
    f = g-h;            60   Else:    sub   $s0, $s1, $s2
End:                    64   End:
```

Example

Convert to assembly:

```
while   (save[i] == k)
    i += 1;
```

Values of i and k are in $s3 and $s5 and base of array save[] is in $s6

Example

lw reg Save[i]

what is addr of save[i] ?

Convert to assembly:

while (save[i] == k)
    i += 1;

exit loop when not equal

$t1 = $s6 + 4 \times $s3

Values of i and k are in $s3 and $s5 and base of array save[] is in $s6

addr of save[i] = baseaddr + 4i

```
Loop: sll   $t1, $s3, 2
      add   $t1, $t1, $s6
      lw    $t0, 0($t1)
      bne   $t0, $s5, Exit
      addi  $s3, $s3, 1
      j     Loop
Exit:
```

← 60 instrs

```
      sll   $t1, $s3, 2
      add   $t1, $t1, $s6
Loop: lw    $t0, 0($t1)
      bne   $t0, $s5, Exit
      addi  $s3, $s3, 1
      addi  $t1, $t1, 4
      j     Loop
Exit:
```

incr i

52 instrs

$t1 will have addr of save[i]

Registers

- The 32 MIPS registers are partitioned as follows:

$1 - assembler

  - Register 0 : $zero always stores the constant 0
  - Regs 2-3 : $v0, $v1 return values of a procedure
  - Regs 4-7 : $a0-$a3 input arguments to a procedure
  - Regs 8-15 : $t0-$t7 temporaries
  - Regs 16-23: $s0-$s7 variables
  - Regs 24-25: $t8-$t9 more temporaries
  - Reg 28 : $gp global pointer
  - Reg 29 : $sp stack pointer
  - Reg 30 : $fp frame pointer
  - Reg 31 : $ra return address

$26, 27 - kernel

Procedures

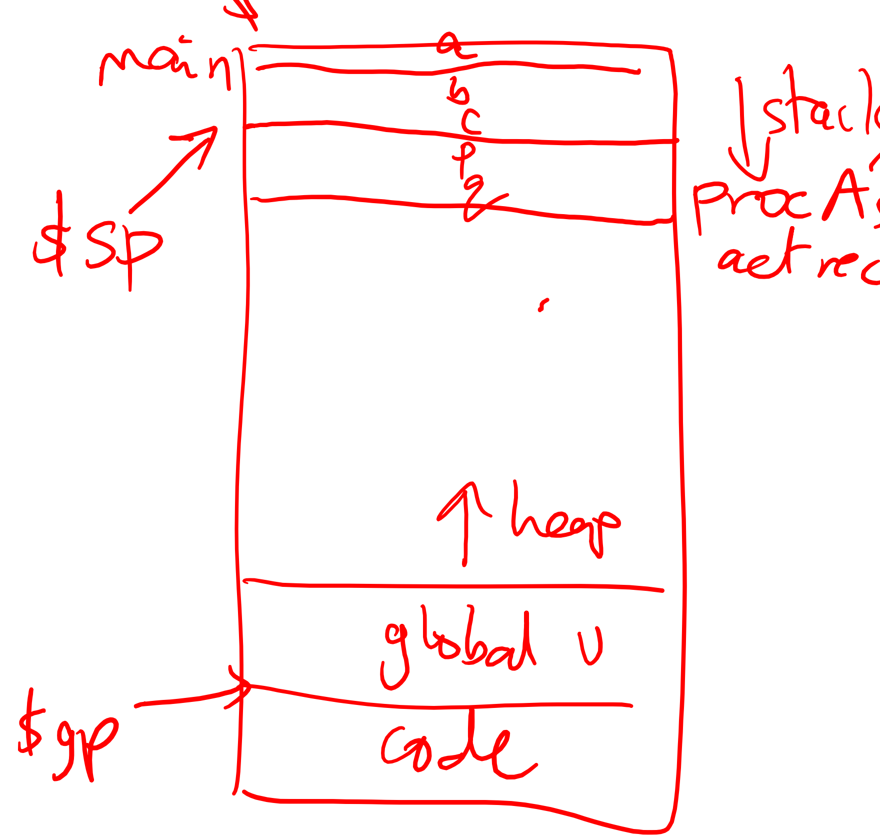
$fp act record begins

PC = 0
main
int a, b, c

procA

procB

Act record



PC = 3000
procA ( )
{ int p, q
procB( )
}

PC = 5000
procB( )
{
}

- Local variables, AR, $fp, $sp
- Scratchpad and saves/restores
- Arguments and returns
- jal and $ra

Procedures

- Local variables, AR, $fp, $sp
- Scratchpad and saves/restores
- Arguments and returns
- jal and $ra