

Lecture 7: Examples, MARS

- Today's topics:
 - More examples
 - MARS intro

Example 2 (pg. 101)

```
int fact (int n)
{
    if (n < 1) return (1);
    else return (n * fact(n-1));
}
```

Notes:

The caller saves \$a0 and \$ra
in its stack space.

Temp register \$t0 is never saved.

```
fact:
    slti    $t0, $a0, 1
    beq    $t0, $zero, L1
    addi   $v0, $zero, 1
    jr     $ra
L1:
    addi   $sp, $sp, -8
    sw     $ra, 4($sp)
    sw     $a0, 0($sp)
    addi   $a0, $a0, -1
    jal    fact
    lw     $a0, 0($sp)
    lw     $ra, 4($sp)
    addi   $sp, $sp, 8
    mul    $v0, $a0, $v0
    jr     $ra
```

Example 2 (pg. 101)

```
int fact (int n)
{
    if (n < 1) return (1);
    else return (n * fact(n-1));
}
```

Notes:

The caller saves \$a0 and \$ra
in its stack space.

Temp register \$t0 is never saved.

```
fact:
    slti    $t0, $a0, 1
    beq    $t0, $zero, L1
    addi   $v0, $zero, 1
    jr     $ra
```

L1:

```
    addi   $sp, $sp, -8
    sw     $ra, 4($sp)
    sw     $a0, 0($sp)
    addi   $a0, $a0, -1
    jal    fact
    lw     $a0, 0($sp)
    lw     $ra, 4($sp)
    addi   $sp, $sp, 8
    mul    $v0, $a0, $v0
    jr     $ra
```

\$t0 = (n < 1)
if false, jump
else, return 1

Example 2 (pg. 101)

```
int fact (int n)
{
    if (n < 1) return (1);
    else return (n * fact(n-1));
}
```

Notes:

The caller saves \$a0 and \$ra
in its stack space.

Temp register \$t0 is never saved.

```
fact:
    slti    $t0, $a0, 1
    beq    $t0, $zero, L1
    addi   $v0, $zero, 1
    jr     $ra
L1:
    addi   $sp, $sp, -8
    sw     $ra, 4($sp)
    sw     $a0, 0($sp)
    addi   $a0, $a0, -1
    jal    fact
    lw     $a0, 0($sp)
    lw     $ra, 4($sp)
    addi   $sp, $sp, 8
    mul    $v0, $a0, $v0
    jr     $ra
```

Saving on
the stack

Restoring
from the
stack

Example 2 (pg. 101)

```
int fact (int n)
{
    if (n < 1) return (1);
    else return (n * fact(n-1));
}
```

Notes:

The caller saves \$a0 and \$ra
in its stack space.

Temp register \$t0 is never saved.

```
fact:
    slti    $t0, $a0, 1
    beq    $t0, $zero, L1
    addi   $v0, $zero, 1
    jr     $ra
```

L1:

```
    addi   $sp, $sp, -8
    sw     $ra, 4($sp)
    sw     $a0, 0($sp)
    addi   $a0, $a0, -1
    jal    fact
    lw     $a0, 0($sp)
    lw     $ra, 4($sp)
    addi   $sp, $sp, 8
    mul    $v0, $a0, $v0
    jr     $ra
```

fact(n-1)

return(n*fact(n-1))

Dealing with Characters

- Instructions are also provided to deal with byte-sized and half-word quantities: lb (load-byte), sb, lh, sh
- These data types are most useful when dealing with characters, pixel values, etc.
- C employs ASCII formats to represent characters – each character is represented with 8 bits and a string ends in the null character (corresponding to the 8-bit number 0);
A is 65, a is 97

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Example 3 (pg. 108)

```
Convert to assembly:  
void strcpy (char x[], char y[])  
{  
    int i;  
    i=0;  
    while ((x[i] = y[i]) != '\0')  
        i += 1;  
}
```

Notes:

Temp registers not saved.

```
strcpy:  
    addi    $sp, $sp, -4  
    sw      $s0, 0($sp)  
    add     $s0, $zero, $zero  
L1: add    $t1, $s0, $a1  
    lb     $t2, 0($t1)  
    add    $t3, $s0, $a0  
    sb     $t2, 0($t3)  
    beq    $t2, $zero, L2  
    addi   $s0, $s0, 1  
    j      L1  
L2: lw     $s0, 0($sp)  
    addi   $sp, $sp, 4  
    jr     $ra
```


Full Example – Sort in C (pg. 133)

```
void sort (int v[ ], int n)
{
    int i, j;
    for (i=0; i<n; i+=1) {
        for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) {
            swap (v,j);
        }
    }
}
```

```
void swap (int v[ ], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

- Allocate registers to program variables
- Produce code for the program body
- Preserve registers across procedure invocations

The swap Procedure

- Register allocation: \$a0 and \$a1 for the two arguments, \$t0 for the temp variable – no need for saves and restores as we're not using \$s0-\$s7 and this is a leaf procedure (won't need to re-use \$a0 and \$a1)

```
swap:  sll  $t1, $a1, 2
       add  $t1, $a0, $t1
       lw   $t0, 0($t1)
       lw   $t2, 4($t1)
       sw   $t2, 0($t1)
       sw   $t0, 4($t1)
       jr   $ra
```

```
void swap (int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

The swap Procedure

- Register allocation: \$a0 and \$a1 for the two arguments, \$t0 for the temp variable – no need for saves and restores as we're not using \$s0-\$s7 and this is a leaf procedure (won't need to re-use \$a0 and \$a1)

```
swap:  sll  $t1, $a1, 2
       add  $t1, $a0, $t1
       lw   $t0, 0($t1)
       lw   $t2, 4($t1)
       sw   $t2, 0($t1)
       sw   $t0, 4($t1)
       jr   $ra
```

> calculating
address of v[k]

\$t0 = v[k]
\$t2 = v[k+1]
v[k] = \$t2
v[k+1] = \$t0
return

```
void swap (int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

The sort Procedure

- Register allocation: arguments v and n use \$a0 and \$a1, i and j use \$s0 and \$s1; must save \$a0 and \$a1 before calling the leaf procedure
- The outer for loop looks like this: (note the use of pseudo-instrs)

```
        move  $s0, $zero        # initialize the loop
loopbody1: bge  $s0, $a1, exit1  # will eventually use slt and beq
        ... inner loop ...
        addi  $s0, $s0, 1
        j     loopbody1
exit1:
```

```
for (i=0; i<n; i+=1) {
    for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) {
        swap (v,j);
    }
}
```

The sort Procedure

- The inner for loop looks like this:

```
        addi    $s1, $s0, -1      # initialize the loop
loopbody2: blt    $s1, $zero, exit2 # will eventually use slt and beq
        sll    $t1, $s1, 2
        add    $t2, $a0, $t1
        lw     $t3, 0($t2)
        lw     $t4, 4($t2)
        ble    $t3, $t4, exit2
        ... body of inner loop ...
        addi    $s1, $s1, -1
        j      loopbody2
exit2:
```

```
for (i=0; i<n; i+=1) {
    for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) {
        swap (v,j);
    }
}
```

Saves and Restores

- Since we repeatedly call “swap” with \$a0 and \$a1, we begin “sort” by copying its arguments into \$s2 and \$s3 – must update the rest of the code in “sort” to use \$s2 and \$s3 instead of \$a0 and \$a1
- Must save \$ra at the start of “sort” because it will get over-written when we call “swap”
- Must also save \$s0-\$s3 so we don’t overwrite something that belongs to the procedure that called “sort”

Saves and Restores

```
sort:  addi   $sp, $sp, -20
       sw    $ra, 16($sp)
       sw    $s3, 12($sp)
       sw    $s2, 8($sp)
       sw    $s1, 4($sp)
       sw    $s0, 0($sp)
       move  $s2, $a0
       move  $s3, $a1
       ...
       move  $a0, $s2    # the inner loop body starts here
       move  $a1, $s1
       jal   swap
       ...
exit1: lw    $s0, 0($sp)
       ...
       addi  $sp, $sp, 20
       jr    $ra
```

9 lines of C code → 35 lines of assembly

Completed Sort Procedure

- Take a closer look at these slides later if you'd like

```
swap:    sll    $t1, $a1, 2
         add    $t1, $a0, $t1
         lw     $t0, 0($t1)
         lw     $t2, 4($t1)
         sw     $t2, 0($t1)
         sw     $t0, 4($t1)
         jr     $ra
```

```
sort:    addi   $sp, $sp, -20
         sw     $ra, 16($sp)
         sw     $s3, 12($sp)
         sw     $s2, 8($sp)
         sw     $s1, 4($sp)
         sw     $s0, 0($sp)
         move   $s2, $a0
         move   $s3, $a1
         move   $s0, $zero
loopbody1: bge   $s0, $s3, exit1
         addi   $s1, $s0, -1
loopbody2: blt   $s1, $zero, exit2
         sll   $t1, $s1, 2
         add   $t2, $s2, $t1
         lw    $t3, 0($t2)
         lw    $t4, 4($t2)
         ble   $t3, $t4, exit2
         move  $a0, $s2
         move  $a1, $s1
         jal   swap
         addi  $s1, $s1, -1
         j     loopbody2
exit2:   addi  $s0, $s0, 1
         j     loopbody1
exit1:   lw    $s0, 0($sp)
         lw    $s1, 4($sp)
         lw    $s2, 8($sp)
         lw    $s3, 12($sp)
         lw    $ra, 16($sp)
         addi  $sp, $sp, 20
         jr   $ra
```


MARS

- MARS is a simulator that reads in an assembly program and models its behavior on a MIPS processor
- Note that a “MIPS add instruction” will eventually be converted to an add instruction for the host computer’s architecture – this translation happens under the hood
- To simplify the programmer’s task, it accepts pseudo-instructions, large constants, constants in decimal/hex formats, labels, etc.
- The simulator allows us to inspect register/memory values to confirm that our program is behaving correctly

Pseudo Instructions

Convenient Pseudo Instructions

blt- Branch on Less Than

bgt - Branch on Greater Than

ble - Branch on Less Than Equal

bge - Branch on Greater Than or Equal

li - Load Immediate

subi - Subtract Immediate

move - Move

Pseudo Instructions

Check MARS
documentation/help
for more
information on
pseudo instructions



MARS 4.5 Help

MIPS MARS License Bugs/Comments Acknowledgements Instruction Set Song

Basic Instructions Extended (pseudo) Instructions Directives Syscalls Exceptions Macros

abs \$t1,\$t2	ABSolute value : Set \$t1 to absolute value of \$t2 (algorithm from Hacker's Deligh
add \$t1,\$t2,-100	ADDITION : set \$t1 to (\$t2 plus 16-bit immediate)
add \$t1,\$t2,100000	ADDITION : set \$t1 to (\$t2 plus 32-bit immediate)
addi \$t1,\$t2,100000	ADDITION Immediate : set \$t1 to (\$t2 plus 32-bit immediate)
addiu \$t1,\$t2,100000	ADDITION Immediate Unsigned: set \$t1 to (\$t2 plus 32-bit immediate), no overflow
addu \$t1,\$t2,100000	ADDITION Unsigned : set \$t1 to (\$t2 plus 32-bit immediate), no overflow
and \$t1,\$t2,100	AND : set \$t1 to (\$t2 bitwise-AND 16-bit unsigned immediate)
and \$t1,100	AND : set \$t1 to (\$t1 bitwise-AND 16-bit unsigned immediate)
andi \$t1,\$t2,100000	AND Immediate : set \$t1 to (\$t2 bitwise-AND 32-bit immediate)
andi \$t1,100	AND Immediate : set \$t1 to (\$t1 bitwise-AND 16-bit unsigned immediate)
andi \$t1,100000	AND Immediate : set \$t1 to (\$t1 bitwise-AND 32-bit immediate)
b label	Branch : Branch to statement at label unconditionally
beq \$t1,-100,label	Branch if Equal : Branch to statement at label if \$t1 is equal to 16-bit immediat
beq \$t1,100000,label	Branch if Equal : Branch to statement at label if \$t1 is equal to 32-bit immediat
beqz \$t1,label	Branch if Equal Zero : Branch to statement at label if \$t1 is equal to zero
bge \$t1,\$t2,label	Branch if Greater or Equal : Branch to statement at label if \$t1 is greater or eq
bge \$t1,-100,label	Branch if Greater or Equal : Branch to statement at label if \$t1 is greater or eq
bge \$t1,100000,label	Branch if Greater or Equal : Branch to statement at label if \$t1 is greater or eq
bgeu \$t1,\$t2,label	Branch if Greater or Equal Unsigned : Branch to statement at label if \$t1 is grea
bgeu \$t1,-100,label	Branch if Greater or Equal Unsigned : Branch to statement at label if \$t1 is grea
bgeu \$t1,100000,label	Branch if Greater or Equal Unsigned : Branch to statement at label if \$t1 is grea
bgt \$t1,\$t2,label	Branch if Greater Than : Branch to statement at label if \$t1 is greater than \$t2
bgt \$t1,-100,label	Branch if Greater Than : Branch to statement at label if \$t1 is greater than 16-b
bgt \$t1,100000,label	Branch if Greater Than : Branch to statement at label if \$t1 is greater than 32-b
bgtu \$t1,\$t2,label	Branch if Greater Than Unsigned: Branch to statement at label if \$t1 is greater t
bgtu \$t1,-100,label	Branch if Greater Than Unsigned: Branch to statement at label if \$t1 is greater t
bgtu \$t1,100000,label	Branch if Greater Than Unsigned: Branch to statement at label if \$t1 is greater t
ble \$t1,\$t2,label	Branch if Less or Equal : Branch to statement at label if \$t1 is less than or equ
ble \$t1,-100,label	Branch if Less or Equal : Branch to statement at label if \$t1 is less than or equ
ble \$t1,100000,label	Branch if Less or Equal : Branch to statement at label if \$t1 is less than or equ
bleu \$t1,\$t2,label	Branch if Less or Equal Unsigned : Branch to statement at label if \$t1 is less th
bleu \$t1,-100,label	Branch if Less or Equal Unsigned : Branch to statement at label if \$t1 is less th
bleu \$t1,100000,label	Branch if Less or Equal Unsigned : Branch to statement at label if \$t1 is less th
blt \$t1,\$t2,label	Branch if Less Than : Branch to statement at label if \$t1 is less than \$t2
blt \$t1,-100,label	Branch if Less Than : Branch to statement at label if \$t1 is less than 16-bit imm
blt \$t1,100000,label	Branch if Less Than : Branch to statement at label if \$t1 is less than 32-bit imm
bltu \$t1,\$t2,label	Branch if Less Than Unsigned : Branch to statement at label if \$t1 is less than \$
bltu \$t1,-100,label	Branch if Less Than Unsigned : Branch to statement at label if \$t1 is less than 1
bltu \$t1,100000,label	Branch if Less Than Unsigned : Branch to statement at label if \$t1 is less than 3
bne \$t1,-100,label	Branch if Not Equal : Branch to statement at label if \$t1 is not equal to 16-bit
bne \$t1,100000,label	Branch if Not Equal : Branch to statement at label if \$t1 is not equal to 32-bit
bnez \$t1,label	Branch if Not Equal Zero : Branch to statement at label if \$t1 is not equal to z
div \$t1,\$t2,\$t3	DIVision : Set \$t1 to (\$t2 divided by \$t3, integer division)
div \$t1,\$t2,-100	DIVision : Set \$t1 to (\$t2 divided by 16-bit immediate, integer division)
div \$t1,\$t2,100000	DIVision : Set \$t1 to (\$t2 divided by 32-bit immediate, integer division)
divu \$t1,\$t2,\$t3	DIVision Unsigned : Set \$t1 to (\$t2 divided by \$t3, unsigned integer division)
divu \$t1,\$t2,-100	DIVision Unsigned : Set \$t1 to (\$t2 divided by 16-bit immediate, unsigned inte
divu \$t1,\$t2,100000	DIVision Unsigned : Set \$t1 to (\$t2 divided by 32-bit immediate, unsigned inte
l.d \$f2,(\$t2)	load floating point Double precision : Set \$f2 and \$f3 register pair to 64-bit va

Close

slti

```
if ($a0 < 1)
then ...
else ...
```

Easier to implement with
pseudo-instructions like blt, bge.

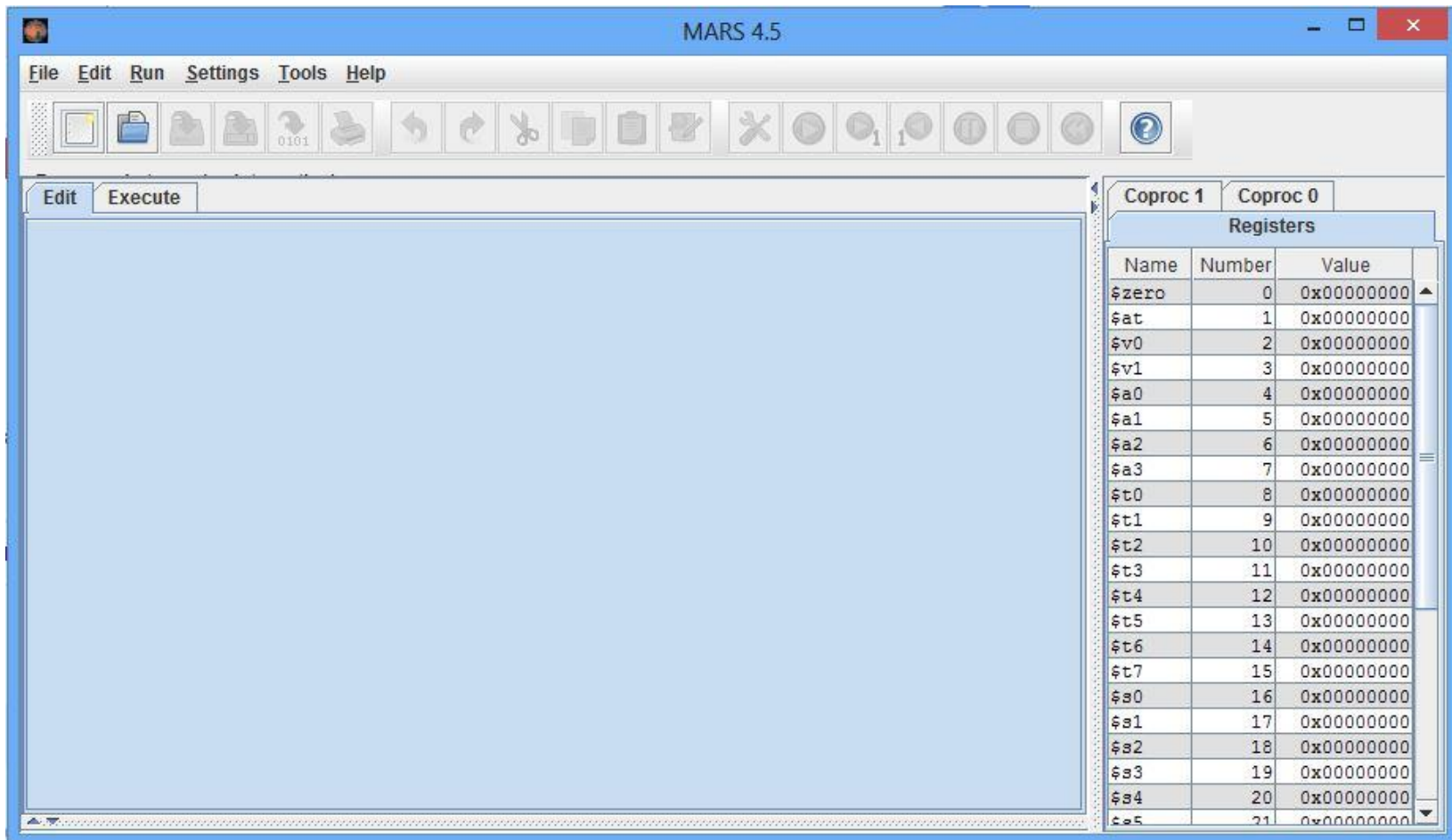
```
slti  $t0, $a0, 1
beq   $t0, $zero, else
then:
    ...
```

if $\$a0 < 1$, set $\$t0 = 1$, else $\$t0 = 0$

```
else:
```

MARS Intro – Tool Bar

- Directives, labels, global pointers, system calls



MARS Intro – Execute Information

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	4194304	0x3c011001	lui \$1,4097	10: la \$s0, list # access the list address
<input type="checkbox"/>	4194308	0x34300000	ori \$16,\$1,0	
<input type="checkbox"/>	4194312	0x24110000	addiu \$17,\$0,0	11: li \$s1, 0 # initialize i
<input type="checkbox"/>	4194316	0x24120028	addiu \$18,\$0,40	12: li \$s2, 40 # search limit
<input type="checkbox"/>	4194320	0x3c011001	lui \$1,4097	14: la \$a0, prompt # load string
<input type="checkbox"/>	4194324	0x34240028	ori \$4,\$1,40	
<input type="checkbox"/>	4194328	0x24020004	addiu \$2,\$0,4	15: li \$v0, 4 # print string
<input type="checkbox"/>	4194332	0x0000000c	syscall	16: syscall
<input type="checkbox"/>	4194336	0x24020005	addiu \$2,\$0,5	18: li \$v0, 5 # User input number syscall
<input type="checkbox"/>	4194340	0x0000000c	syscall	19: syscall
<input type="checkbox"/>	4194344	0x1232000e	beq \$17,\$18,14	22: beq \$s1, \$s2, notfound # break if searching past limit
<input type="checkbox"/>	4194348	0x02115020	add \$10,\$16,\$17	23: add \$t2, \$s0, \$s1 # get address of array item
<input type="checkbox"/>	4194352	0x8d480000	lw \$8,0(\$10)	24: lw \$t0, 0(\$t2) # get array item
<input type="checkbox"/>	4194356	0x11020002	beq \$8,\$2,2	25: beq \$t0, \$v0, found # if number is found in array break

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 \b
268501024	\0 \0 \0 \t	\0 \0 \0 \n	e t n E	a r	b m u n	y r e	w u o	t n a
268501056	f o t	: d n i	o F \0	d n u	b m u n	: r e	d i D \0	t o n
268501088	n i f	u n d	r e b m	\0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501120	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501152	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501184	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501216	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501248	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501280	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501312	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

← →
0x10010000 (.data) ▾
 Hexadecimal Addresses
 Hexadecimal Values
 ASCII

MARS Intro – Data/Memory Information

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 \b
268501024	\0 \0 \0 \t	\0 \0 \0 \n	e t n E	a r	b m u n	y r e	w u o	t n a
268501056	f o t	: d n i	o F \0	d n u	b m u n	: r e	d i D \0	t o n
268501088	n i f	u n d	r e b m	\0 \0 \0 .	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501120	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501152	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501184	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501216	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501248	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501280	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
268501312	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

← → 0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

`.data`

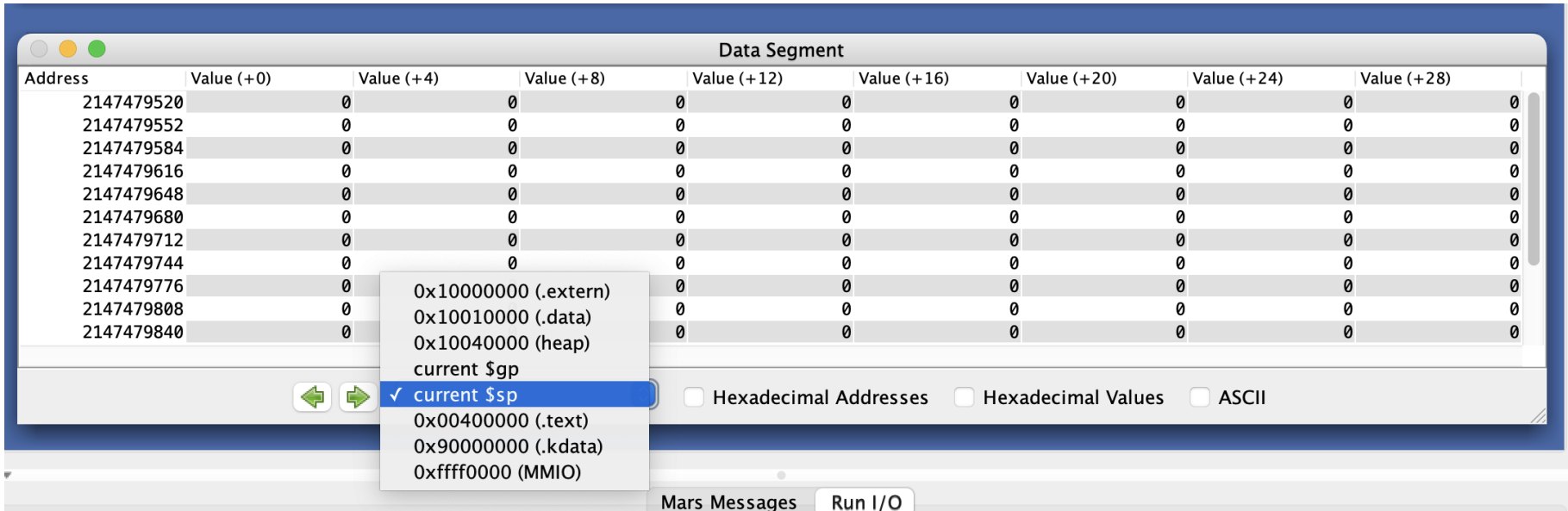
`list: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10`

`prompt: .asciiz "Enter a number you want to find: "`

`found_prompt: .asciiz "Found number: "`

`not_found_prompt: .asciiz "Did not find number."|`

MARS Intro – Data/Memory Information



The screenshot shows the 'Data Segment' window in MARS. The window displays a table of memory addresses and their values. The table has columns for 'Address' and 'Value (+0)' through 'Value (+28)'. The values are all 0. A dropdown menu is open over the 'current \$gp' entry, listing various memory segments: 0x10000000 (.extern), 0x10010000 (.data), 0x10040000 (heap), current \$gp, current \$sp, 0x00400000 (.text), 0x90000000 (.kdata), and 0xffff0000 (MMIO). The 'current \$sp' entry is selected. Below the table, there are checkboxes for 'Hexadecimal Addresses', 'Hexadecimal Values', and 'ASCII'. At the bottom, there are buttons for 'Mars Messages' and 'Run I/O'.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2147479520	0	0	0	0	0	0	0	0
2147479552	0	0	0	0	0	0	0	0
2147479584	0	0	0	0	0	0	0	0
2147479616	0	0	0	0	0	0	0	0
2147479648	0	0	0	0	0	0	0	0
2147479680	0	0	0	0	0	0	0	0
2147479712	0	0	0	0	0	0	0	0
2147479744	0	0	0	0	0	0	0	0
2147479776	0	0	0	0	0	0	0	0
2147479808	0	0	0	0	0	0	0	0
2147479840	0	0	0	0	0	0	0	0

- 0x10000000 (.extern)
- 0x10010000 (.data)
- 0x10040000 (heap)
- current \$gp
- ✓ current \$sp
- 0x00400000 (.text)
- 0x90000000 (.kdata)
- 0xffff0000 (MMIO)

Hexadecimal Addresses Hexadecimal Values ASCII

Mars Messages Run I/O

Debugging with memory values is very useful!

MARS Intro – Register Information

Registers		
Coproc 1		
Coproc 0		
Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	5
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	5
\$t1	9	0
\$t2	10	268501008
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	268500992
\$s1	17	16
\$s2	18	0
\$s3	19	40
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194404
hi		0
lo		0

- 5 in \$t0
- Address in \$s0
- 10 in \$v0
- Incremented address in \$t2
- Read the google doc on the class webpage for details!

System Calls

Print Integer

- move \$a0, \$t0
- li \$v0, 1
- syscall

Print String

- la \$a0, prompt
- li \$v0, 4
- syscall

Exit Program

- li \$v0, 10
- syscall

Table of Available Services

Service	Code in \$v0	Arguments	Result
print integer	1	\$a0 = integer to print	
print float	2	\$f12 = float to print	
print double	3	\$f12 = double to print	
print string	4	\$a0 = address of null-terminated string to print	
read integer	5		\$v0 contains integer read
read float	6		\$f0 contains float read
read double	7		\$f0 contains double read
read string	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read	<i>See note below table</i>
sbrk (allocate heap memory)	9	\$a0 = number of bytes to allocate	\$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	\$a0 = character to print	<i>See note below table</i>
read character	12		\$v0 contains character read
open file	13	\$a0 = address of null-terminated string containing filename \$a1 = flags \$a2 = mode	\$v0 contains file descriptor (negative if error). <i>See note below table</i>
read from file	14	\$a0 = file descriptor \$a1 = address of input buffer \$a2 = maximum number of	\$v0 contains number of characters read (0 if end-of-file, negative if error). <i>See note below table</i>

Example Print Routine

```
.data
str:  .asciiiz "the answer is "
.text
li   $v0, 4      # load immediate; 4 is the code for print_string
la   $a0, str    # the print_string syscall expects the string
                # address as the argument; la is the instruction
                # to load the address of the operand (str)
syscall          # MARS will now invoke syscall-4
li   $v0, 1      # syscall-1 corresponds to print_int
li   $a0, 5      # print_int expects the integer as its argument
syscall          # MARS will now invoke syscall-1
```

Example

- Write an assembly program to prompt the user for two numbers and print the sum of the two numbers

Example

.text

```
li $v0, 4
la $a0, str1
syscall
li $v0, 5
syscall
add $t0, $v0, $zero
li $v0, 5
syscall
add $t1, $v0, $zero
li $v0, 4
la $a0, str2
syscall
li $v0, 1
add $a0, $t1, $t0
syscall
```

.data

```
str1: .ascii "Enter 2 numbers:"
str2: .ascii "The sum is "
```