# SCC150 – MIPS/Assembly Week 14/16 Assessed Practical

Angie Chandler
SCC – Lancaster University
Week 14/16

# Extra Tips

- This set of slides contains any additional tips you might need
- Syscall
- Conditional branch
- Interpret instruction
- Drawing on screen
- Memory layout
- Drawing lines
- Memory layout
- CLS implementation
- Procedures
- Logo
- Marking yourself

# Coursework goals – Lecture Refs

1. CLI interface using syscalls (topic 7) and loops (topic 2) to read user requirements.
2. Paint pixels in bitmap using memory access (topic 2).
3. You do not need to write a procedure for cls.
4. Implement a draw_line function (topic 6).
   - Input: a1 -  row or column flag, a1 - row or column location.
5. Implement a logo function (topic 6).
   - Input: a0 - start point in memory, a2 - direction, a3 - length in pixels.
   - Return: v0 – end point.

# SYSCALL Execution

1. Load the service number in register $v0
2. Load argument values, if any, in $a0, $a1, $a2, or $f12
3. Issue the SYSCALL instruction
4. Retrieve return values, if any, from result registers

Note: MIPS register contents are not affected by a system call, except for result registers as specified.

# Syscall Service Numbers

| Service | System call code | Arguments | Result |
|---|---|---|---|
| print_int | 1 | $a0=integer | |
| print_float | 2 | $f12=float | |
| print_double | 3 | $f12=double | |
| print_string | 4 | $a0=string | |
| read_int | 5 | | integer ($v0) |
| read_float | 6 | | float ($f0) |
| read_double | 7 | | double ($f0) |
| read_string | 8 | $a0=buffer, $a1=length | |
| sbrk | 9 | $a0=amount | |
| exit | 10 | | |

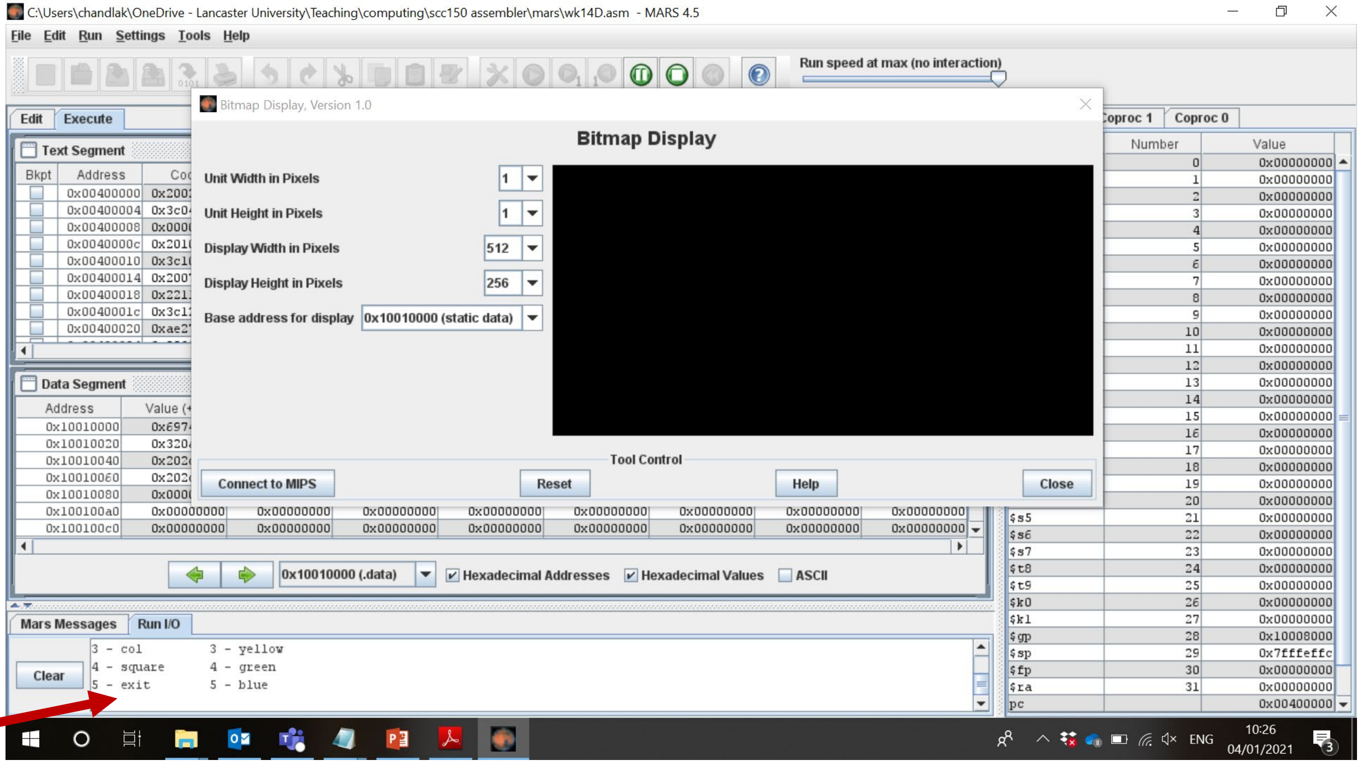
# Syscall - Printing a String

```
.data
#using .data to store the string you want to print
#.asciiz tells it that it has an ascii string and that it ends with a zero
#info is just a label
info:     .asciiz            “This string has instructions”

.text
#back to main program here (.text)
          addi $v0,$zero,4                #tell syscall to print string (put code 4 into $v0)
          la $a0, info                     #load the string address (label info) into $a0
                                           #$a0 is the parameter for syscall
          syscall                          #call syscall – it will only print now
```

# Syscall - Reading an Integer

- To read an integer, you must put code 5 into syscall ($v0)
- When you call syscall, you can type the integer in using the bottom window
- The result is stored in $v0



C:\Users\chandlak\OneDrive - Lancaster University\Teaching\computing\scc150 assembler\mars\wk14D.asm - MARS 4.5

Text Segment

| Bkpt | Address | Coc |
|---|---|---|
| □ | 0x00400000 | 0x200 |
| □ | 0x00400004 | 0x3c0 |
| □ | 0x00400008 | 0x000 |
| □ | 0x0040000c | 0x201 |
| □ | 0x00400010 | 0x3c1 |
| □ | 0x00400014 | 0x200 |
| □ | 0x00400018 | 0x221 |
| □ | 0x0040001c | 0x3c1 |
| □ | 0x00400020 | 0xae2 |

Data Segment

| Address | Value (+ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x10010000 | 0x697 | | | | | | | |
| 0x10010020 | 0x320 | | | | | | | |
| 0x10010040 | 0x202 | | | | | | | |
| 0x10010060 | 0x202 | | | | | | | |
| 0x10010080 | 0x000 | | | | | | | |
| 0x100100a0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x100100c0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

Bitmap Display, Version 1.0

Bitmap Display

Unit Width in Pixels: 1

Unit Height in Pixels: 1

Display Width in Pixels: 512

Display Height in Pixels: 256

Base address for display: 0x10010000 (static data)

Tool Control

| | Number | Value |
|---|---|---|
| | 0 | 0x00000000 |
| | 1 | 0x00000000 |
| | 2 | 0x00000000 |
| | 3 | 0x00000000 |
| | 4 | 0x00000000 |
| | 5 | 0x00000000 |
| | 6 | 0x00000000 |
| | 7 | 0x00000000 |
| | 8 | 0x00000000 |
| | 9 | 0x00000000 |
| | 10 | 0x00000000 |
| | 11 | 0x00000000 |
| | 12 | 0x00000000 |
| | 13 | 0x00000000 |
| | 14 | 0x00000000 |
| | 15 | 0x00000000 |
| | 16 | 0x00000000 |
| | 17 | 0x00000000 |
| | 18 | 0x00000000 |
| | 19 | 0x00000000 |
| | 20 | 0x00000000 |
| $s5 | 21 | 0x00000000 |
| $s6 | 22 | 0x00000000 |
| $s7 | 23 | 0x00000000 |
| $t8 | 24 | 0x00000000 |
| $t9 | 25 | 0x00000000 |
| $k0 | 26 | 0x00000000 |
| $k1 | 27 | 0x00000000 |
| $gp | 28 | 0x10008000 |
| $sp | 29 | 0x7fffeffc |
| $fp | 30 | 0x00000000 |
| $ra | 31 | 0x00000000 |
| pc | | 0x00400000 |

Run I/O:

```
3 - col          3 - yellow
4 - square       4 - green
5 - exit         5 - blue
```

here

# Conditional Branch

- Different conditions can be tested
  - branch on equal: beq
  - branch on not equal: bne
  - set on less than: slt (used to support beq and bne)
- PC-relative addressing
  - take the address offset and sign extend to 32 bits
  - multiply by four (shift left by two places)
  - add to the program counter
  - 16-bit PC-relative
    - 2^16 = 65536 words
    - -32768 or +32767 instr. away from PC

# Branch Example
# For loop

```
for (i = 0; i < 10; i++)
    j++;
```

```
      addi $s0, $zero, 0
loop: bne  $s0,10, Exit
      addi $s1,$s1,1
      addi $s0, $s0, 1
      j    loop
exit:
```

register mapping

i: $s0

j: $s1

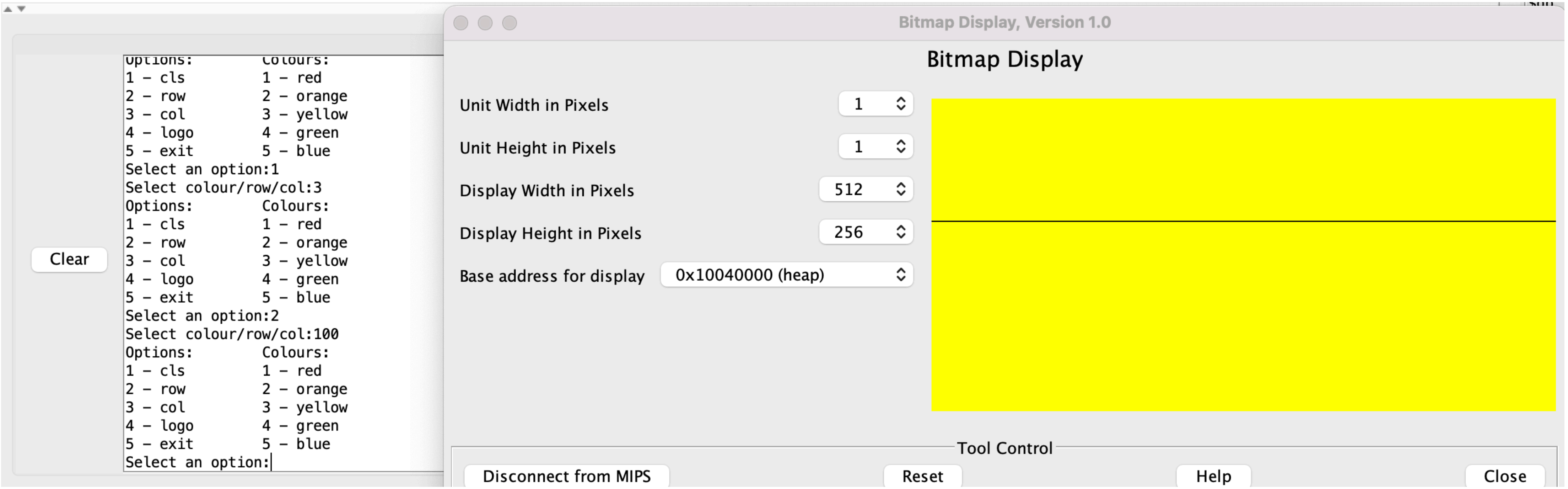
# Interpret Instruction

- You’ve told the user what to do
- You’ve read the integer
- Decide what to do with the instruction
- There are two types of instruction:
- Select task
  1. cls (colour background)
  2. Row
  3. Column
  4. Logo (dynamic drawing)
  5. Exit
- Select colour – there should be several choices of colour

# Drawing on screen

- You must be able to carry out the instruction
- Week 12’s practical instructions will help with this if you’ve forgotten
- CLS
  - Fill in the background of the screen in the correct colour
- Row
  - Draw a horizontal line (in the correct colour)
- Column
  - Draw a vertical line (in the correct colour)
- Logo
  - Use N, S, E, W (north, south, east, west) and distance. You should use a char input to read direction.
- Exit

# Example – cls, row



```
Options:        Colours:
1 - cls         1 - red
2 - row         2 - orange
3 - col         3 - yellow
4 - logo        4 - green
5 - exit        5 - blue
Select an option:1
Select colour/row/col:3
Options:        Colours:
1 - cls         1 - red
2 - row         2 - orange
3 - col         3 - yellow
4 - logo        4 - green
5 - exit        5 - blue
Select an option:2
Select colour/row/col:100
Options:        Colours:
1 - cls         1 - red
2 - row         2 - orange
3 - col         3 - yellow
4 - logo        4 - green
5 - exit        5 - blue
Select an option:
```

Bitmap Display, Version 1.0

Bitmap Display

Unit Width in Pixels: 1

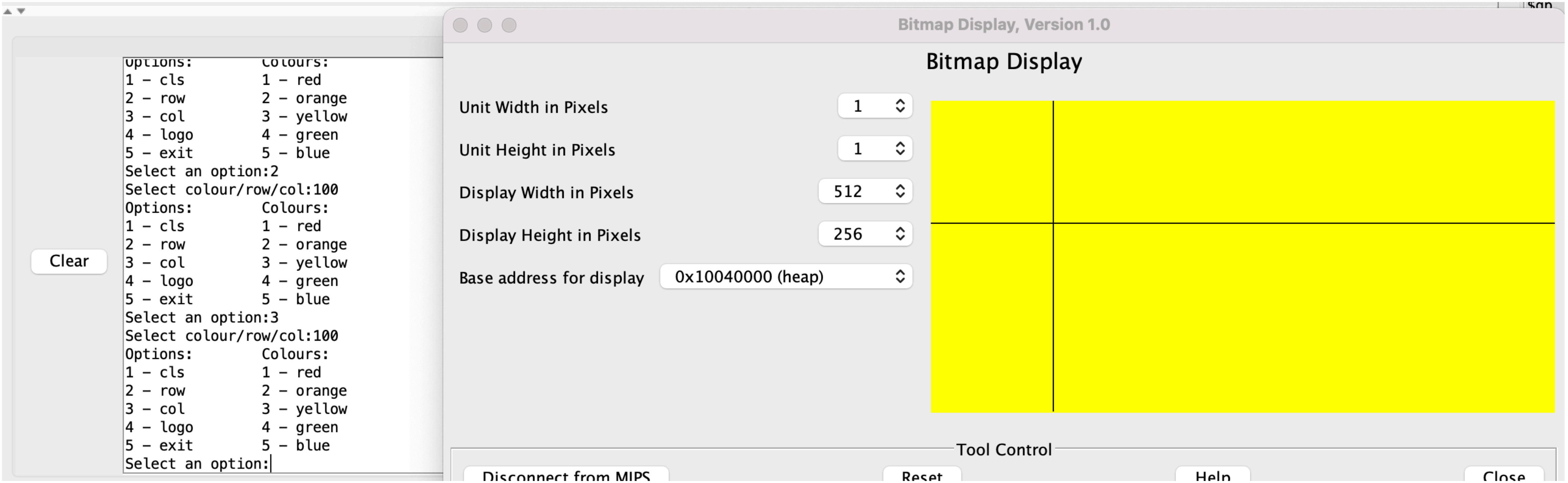
Unit Height in Pixels: 1

Display Width in Pixels: 512

Display Height in Pixels: 256

Base address for display: 0x10040000 (heap)

# Example - column



```
Options:         Colours:
1 – cls          1 – red
2 – row          2 – orange
3 – col          3 – yellow
4 – logo         4 – green
5 – exit         5 – blue
Select an option:2
Select colour/row/col:100
Options:         Colours:
1 – cls          1 – red
2 – row          2 – orange
3 – col          3 – yellow
4 – logo         4 – green
5 – exit         5 – blue
Select an option:3
Select colour/row/col:100
Options:         Colours:
1 – cls          1 – red
2 – row          2 – orange
3 – col          3 – yellow
4 – logo         4 – green
5 – exit         5 – blue
Select an option:
```

Bitmap Display, Version 1.0

Bitmap Display

Unit Width in Pixels: 1

Unit Height in Pixels: 1

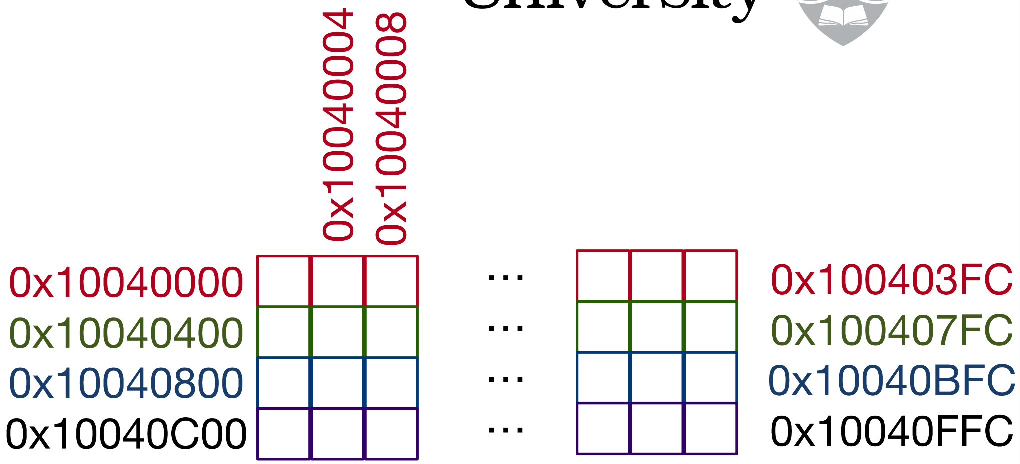
Display Width in Pixels: 512

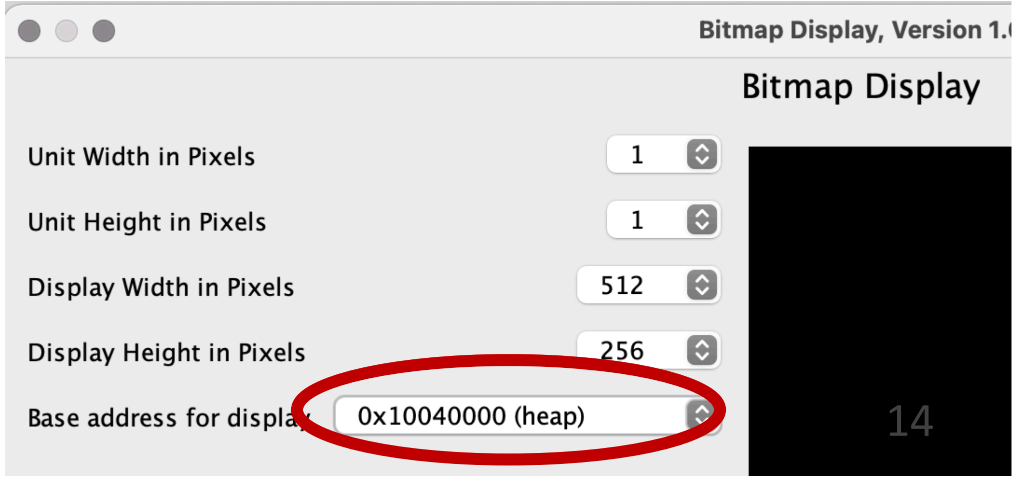
Display Height in Pixels: 256

Base address for display: 0x10040000 (heap)

# Memory layout

- Each pixel is a word; 0x00000000
- Pixel orientation: left to right, top to bottom.
  - (0, 0) → 0x10040000
  - (1, 0) → 0x10040000 + 1*4
  - (2, 0) → 0x10040000 + 2*4
  - (0,1) → 0x10004000 + 0*4 + 1*265*4
  - (1, 2) → 0x10004000 + 1*4 + 2*265*4





Bitmap Display

Unit Width in Pixels 1

Unit Height in Pixels 1

Display Width in Pixels 512

Display Height in Pixels 256

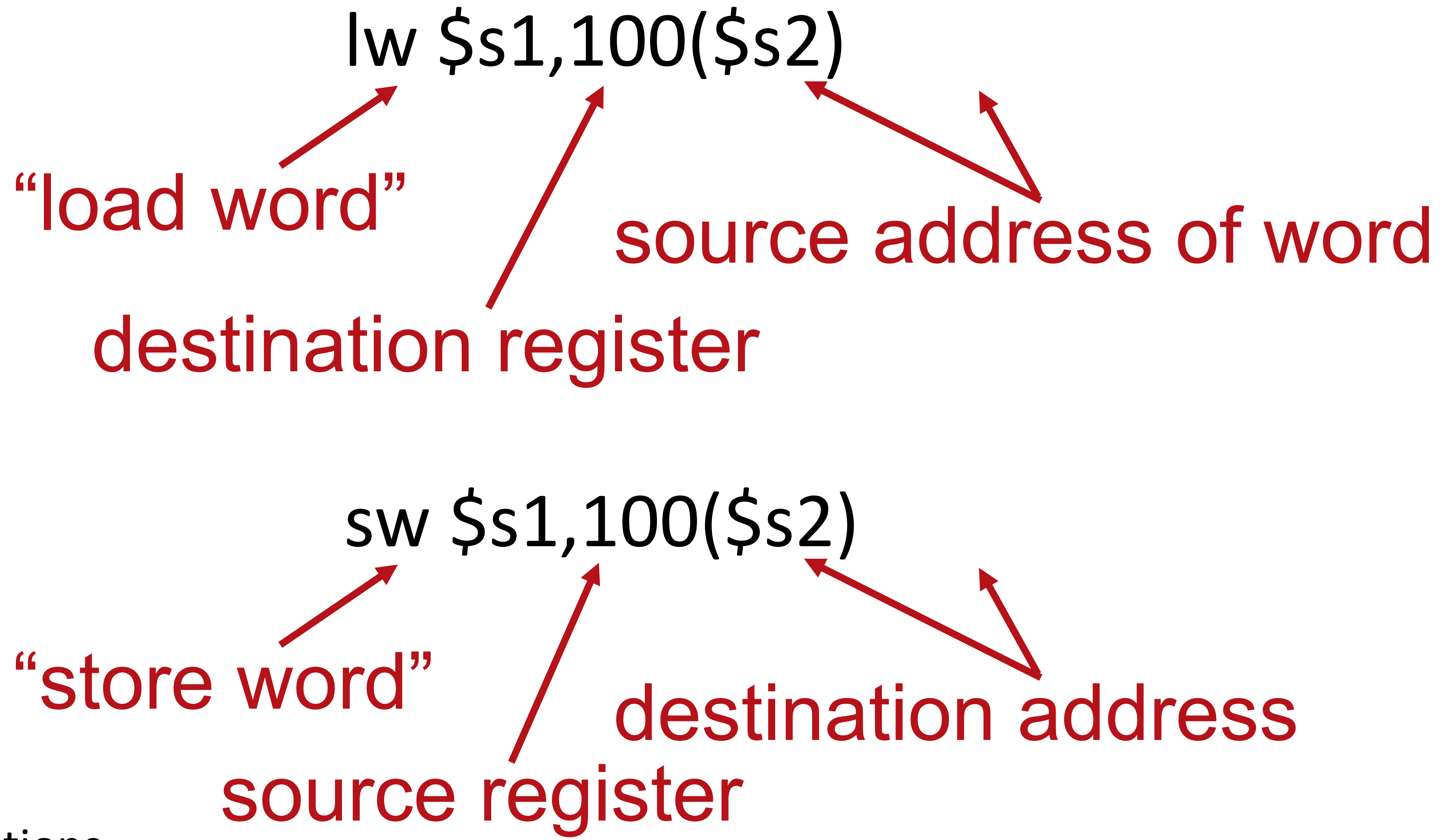
Base address for display 0x10040000 (heap)

**NOTE**: MIPS mult instruction requires access to a co-processor.
We suggest the use of loops and additions to multiply.

# Drawing lines - algorithm

1. Find start point and set in memory pointer register.
2. Paint pixel using a memory write.
3. Add offset to memory pointer register to find the memory location of the next pixel.
4. If we have not reached the end of the line, go to 2.

# Accessing Memory



lw $s1,100($s2)

“load word”

destination register

source address of word

sw $s1,100($s2)

“store word”

source register

destination address

lw,sw are **I-type** Instructions

# CLS – Example (in blue)

```
lui $s0,0x1004                    #bitmap display base address in $s0
addi $t8,$zero,0x00ff             #set colour to blue in $t8
addi $t0,$s0,0                    #initialise $t0 to base address, will count
lui $s1,0x100C                    #end of screen area in $s1

drawPixel:                              #label
        sw $t8,0($t0)                   #store colour $t8 in current target address
        addi $t0,$t0,4            #increment $t0 by one word
        bne $t0,$s1,drawPixel#if haven’t reached the target yet, repeat
```

# Steps for procedure implementation

1. Place parameters in a place where the procedure has access.
2. Transfer control to the procedure.
3. Acquire the storage resources (e.g. variables, arrays) needed for the procedure.
4. Perform the desired task.
5. Place the result value in a place where the caller can access it.
6. Return control to the point of origin.

# MIPS registers for procedure calls

$a0-$a3:

“argument” registers in which to pass parameters

$v0 and $v1:

return value registers

$ra:

return address register

# MIPS instructions for procedure calls

```
jal ProcedureAddress
```

"jump and link"

label to jump to

- jal stores the address of the next instruction in $ra
- ...and then jumps to ProcedureAddress
- to get back, we use "jump register"

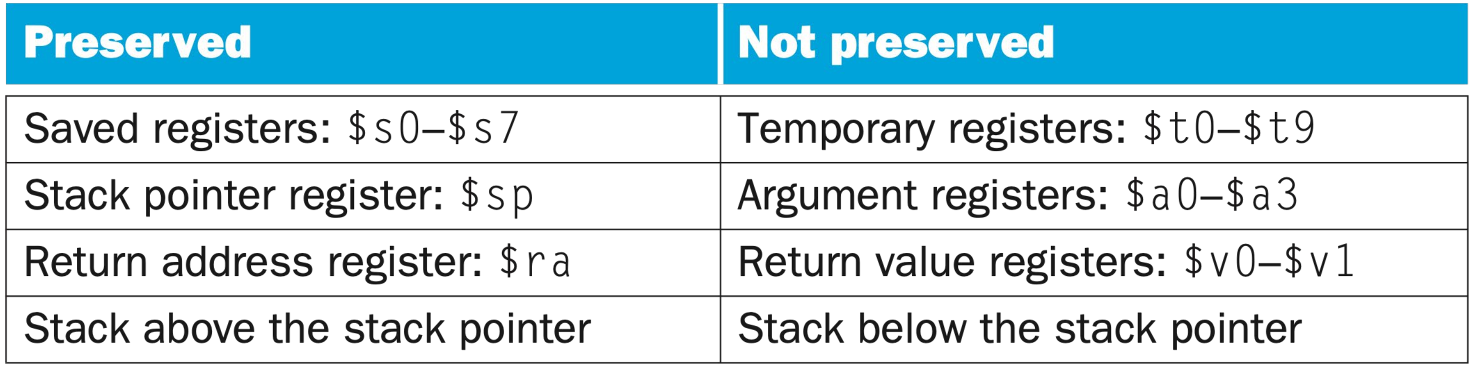
```
jr $ra
```

"jump register"

# Preserving registers – register spilling

- Sometimes a procedure needs to use more registers than just four arguments and two return values.
- Register content must be preserved during procedure call
- Moving the contents of registers to main memory is called **spilling registers**.
- Registers are stored to memory using a conceptual data structure known as a **stack**.
- The stack pointer register $sp points to the contents of the register most recently pushed onto the stack.

# Preserving registers – which registers?



| Preserved | Not preserved |
|---|---|
| Saved registers: $s0–$s7 | Temporary registers: $t0–$t9 |
| Stack pointer register: $sp | Argument registers: $a0–$a3 |
| Return address register: $ra | Return value registers: $v0–$v1 |
| Stack above the stack pointer | Stack below the stack pointer |

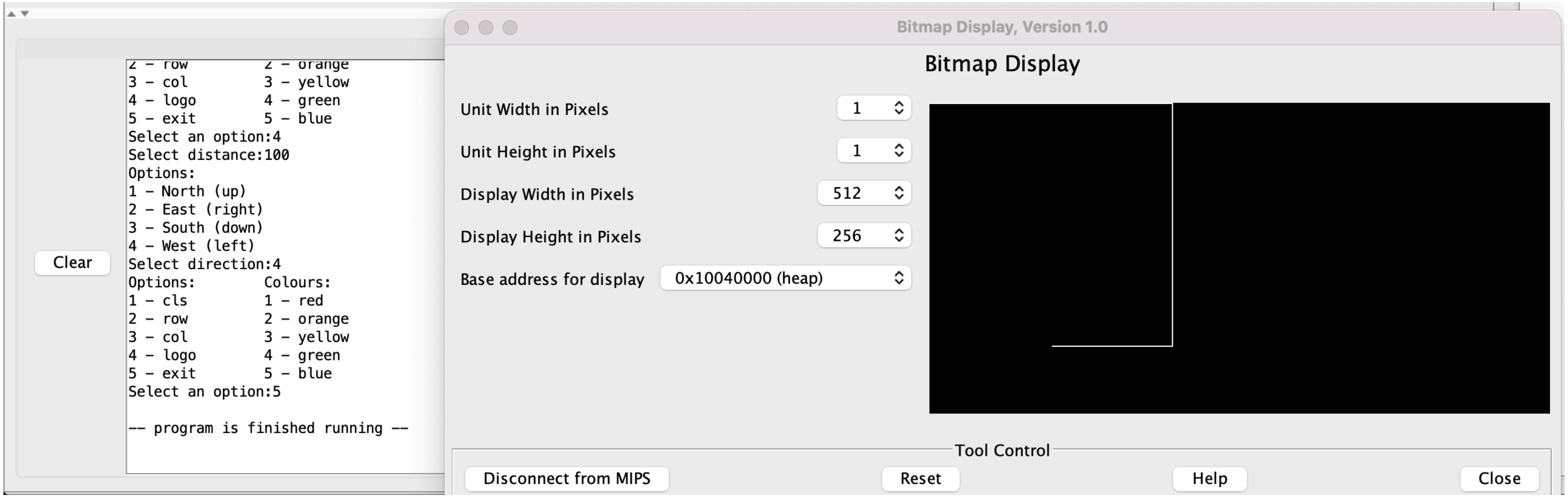
P&H fig. 2.15

- MIPS convention
  - s registers must be restored after procedure call
  - If usage of these registers is avoided no spilling of registers on the stack is required.

# LOGO

- Logo is a drawing program https://www.transum.org/software/Logo/
- For the purpose of this exercise, you need to be able to draw up (North), down (South), left (West) or right (East) on the screen
- You also need to give the cursor a distance to travel
- For this program, you need to select N, S, E, W using syscall read char (code 12 – see list here) http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html

# Example - logo



```
2 – row          2 – orange
3 – col          3 – yellow
4 – logo         4 – green
5 – exit         5 – blue
Select an option:4
Select distance:100
Options:
1 – North (up)
2 – East (right)
3 – South (down)
4 – West (left)
Select direction:4
Options:         Colours:
1 – cls          1 – red
2 – row          2 – orange
3 – col          3 – yellow
4 – logo         4 – green
5 – exit         5 – blue
Select an option:5

–– program is finished running ––
```

Bitmap Display, Version 1.0

Bitmap Display

Unit Width in Pixels: 1

Unit Height in Pixels: 1

Display Width in Pixels: 512

Display Height in Pixels: 256

Base address for display: 0x10040000 (heap)

Input data:

- 4 (logo) – 200 (distance) – 2 (east)
- 4 (logo) – 200 (distance) – 3 (south)
- 4 (logo) – 100 (distance) – 4 (West)

# Marking Yourself

- Normally, marking is done in the lab with one of the TAs
- Part of that process is you telling the TA how you think you’ve done, and them providing you feedback on that
- Using the marking scheme provided, you should note the grades you expect to get, with a VERY BRIEF (a few words – use the marking scheme!) explanation
- This will enable us to provide better feedback
- And it will ensure that we don’t struggle with programs that don’t work