Week 9 Review

W9 Material

- OS services (syscalls)
- Memory Instructions
  - Alignment and Endianess
- Structs and Arrays
- Stack
- Function calls

Question 1a

- Write a piece of code to compute:
  $t2 = max of $t0 and $t1
  - Assume values a,b are already in $t0, $t1

```
# input values are in $t0, $t1, output will be in $t2
      ble $t0,$t1, else        # if a<=b we jump to else
      add $t2, $t0, $zero      # a>b so set $t2 to $t0
      j end
else: add $t2,$t1,$zero        # a<=b so set $t2 to $t1
end:
```

Question 1b

- Convert the previous code to function max(a,b)
  - Get a, b parameters from stack, result in return value

```
# input values are in $t0, $t1, output will be in $t2
      ble $t0,$t1, else        # if a<=b we jump to else
      add $t2, $t0, $zero      # a>b so set $t2 to $t0
      j end
else: add $t2,$t1,$zero        # a<=b so set $t2 to $t1
end:
```

Question 1b

- Convert the previous code to function max(a,b)
  - Get a, b parameters from stack, result in return value

```
max:   lw $t1, 0($sp)          # first pop b from stack
       addi $sp, $sp, 4
       lw $t0, 0($sp)          # now pop a from stack
       addi $sp, $sp, 4

# input values are in $t0, $t1, output will be in $t2
       ble $t0,$t1, else       # if a<=b we jump to else
       add $t2, $t0, $zero     # a>b so set $t2 to $t0
       j end
else:  add $t2,$t1,$zero       # a<=b so set $t2 to $t1
end:   addi $sp, $sp, -4       # push result onto stack
       sw $t2, 0($sp)
       jr $ra                  # jump back to caller
```

Question 1c

- Call the function you just wrote from main!

Question 1c

- Call the function you just wrote from main!

```
main: addi, $t4, $zero, -3    # prepare first value
      addi $sp, $sp, -4       # make space on stack
      sw $t4, 0($sp)          # put on stack
      addi, $t4, $zero, 9     # push second value onto
      addi $sp, $sp, -4       # the stack
      sw $t4, 0($sp)
      jal max                 # "call" the max function
      lw $t4, 0($sp)          # pop the result from the
      addi $sp, $sp, 4        # the stack
      # result is now in $t4
```

Question 2

- Write a sign function

```
def sign(i):
  if(i > 0):
    result = 1
  elif(i < 0):
    result = -1
  else:
    result = 0
  return result
```

Question 3: Implement strcpy

```
int strcpy (char dst[], char src[]) {
   int i;
   i=0;
   while ( (dst[i] = src[i]) != 0 )
      i += 1;
   return 1;
}
```

Passing Arrays to Function?

```
int strcpy (char dst[], char src[]) {
   int i;
   i=0;
   while ( (dst[i] = src[i]) != 0 )
      i += 1;
   return 1;
}
```

- Pass the address of first element of the array!
  - And ideally, length or some way to detect the end.
- This is how it is done in the C language as well!
- In C, strings end with the value `0`.

Converting strcpy()

Initialization:

- Parameters:
  - Addresses of dst and src
- We'll also need registers for:
  - The current offset value (i in this case)
  - Temporary values for the address of dst[i] and src[i]
  - The current value being copied from src[i] to dst[i].

```
int strcpy (char dst[], char src[])
{
   int i;
   i=0;
   while ( (dst[i] = src[i]) != 0 )
      i += 1;
   return 1;
}
```

Converting strcpy()

```
int strcpy (char dst[], char src[]) {
   int i;
   i=0;
   while ( (dst[i] = src[i]) != 0 )
      i += 1;
   return 1;
}
```

- Main algorithm: What steps do we need to perform?
  - Get the location of dst[i] and src[i].
  - Fetch a character from src[i] and store it in dst[i].
  - Jump to the end if the character is the NUL character.
  - Otherwise, increment i and jump to the beginning.
- At the end: push the value 1 onto the stack and return to the calling program.

Translated strcpy program

```
strcpy:          lw      $a0, 0($sp)          # pop src address
                 addi    $sp, $sp, 4          #   off the stack
  initialization lw      $a1, 0($sp)          # pop dst address
                 addi    $sp, $sp, 4          #   off the stack
                 add     $t0, $zero, $zero    # $t0 is offset i
COPY:            add     $t1, $t0, $a0        # $t1 = src + i
                 lb      $t2, 0($t1)          # $t2 = src[i]
main algorithm   add     $t3, $t0, $a1        # $t3 = dst + i
                 sb      $t2, 0($t3)          # dst[i] = $t2
                 beq     $t2, $zero, DONE     # dst[i] = '\0'?
                 addi    $t0, $t0, 1          # i++
                 j       COPY                 # loop back
DONE:            addi    $t0, $zero, 1        # push 1 onto
             end addi    $sp, $sp, -4         #   the top of
                 sw      $t0, 0($sp)          #   the stack
                 jr      $ra                  # return
```

Question 4

- What does the following function do?:

```
myfunc:       lw $t0, 0($sp)
              addi $sp, $sp, 4
              addi $t1, $zero, 2
              div $t0, $t1
              mfhi $t0
              beq $t0, $zero, LABEL1
              add $t2, $zero, $zero
              j LABEL2

LABEL1:       addi $t2, $zero, 1
LABEL2:       addi $sp, $sp, -4
              sw $t2, 0($sp)
              jr $ra
```

Question 4

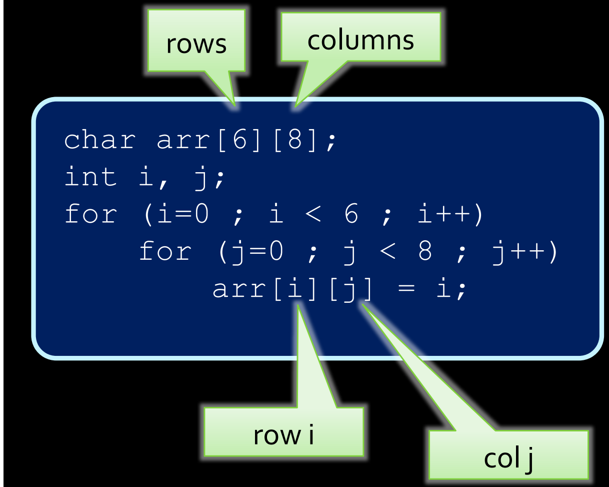
- We divided `$t0` by 2
  - `div` puts remainder in HI
- If remainder is 0 → return 1
- if remainder is not 0 → return 0
- This is a function that returns 1 if a number is even or 0 if it is odd

Question 4b

Now write some code to do the following:

- Create an array of integers
  - The last value in the array is zero (to stop the loop)
- Use the function we just saw to count the number of even values in the array.
- Use $s0-$s7 in main so that the function does not overwrite your registers!
  - We'll learn to deal with it in W10.

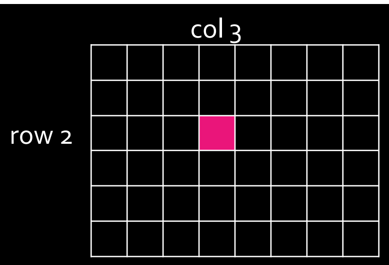
Question 5:
fill 2D array in 4 ways



```
char arr[6][8];
int i, j;
for (i=0 ; i < 6 ; i++)
    for (j=0 ; j < 8 ; j++)
        arr[i][j] = i;
```

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5

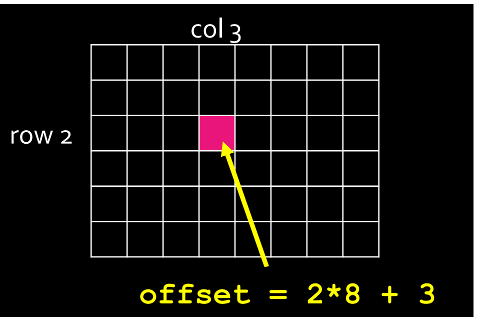
Question 5



offset = ?

- Stop and think.
- Try to answer these first
  1. How do we declare the 2D array in assembly?
  2. How do we access `arr[i][j]`?
  3. How do I do nest loops in assembly?

Question 5



- Stop and think.
- Try to answer these first
  1. How do we declare the 2D array in assembly?
     - You can’t. Declare a 1D array of length rows*columns*size
  2. How do we access `arr[i][j]`?
     - Compute offset → `base + i*ROW_WIDTH + j`
  3. How do I do nest loops in assembly?
     - Assembly doesn’t even understand loops.
     - Have multiple labels and do jumps correctly so there is an inner and outer loop

Question 5

- First declare array

```
.data
arr:   .byte  0:48   # char array with 6 rows and 8 columns
```

- We’ll now see 4 ways to fill it!

Question 5: nested loop

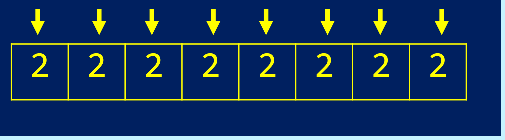
```
        la $t9, arr # $t9 is base address of arr
        li $t0, 0   # $t0 is i
        li $t2, 6   # $t2 = number of rows
        li $t3, 8   # $t3 = number of columns

outer: beq $t0, $t2, end_outer # finish when i == 6

        li $t1, 0   # restart inner loop every iteration
inner: beq $t1, $t3, end_inner # finish j == 8
        sll $t4, $t0, 3   # compute i*8
        add $t4, $t4, $t1 # offset = i*8 + j
        add $t4, $t9, $t4 # address is arr + offset
        sb $t0, 0($t4)    # arr[i][j] = i
        # move to next iteration (inner)
        addi $t1, $t1, 1
        j inner_loop
end_inner:
        # move to next iteration (outer)
        addi $t0, $t0, 1
        j outer_loop
end_outer:
```

Question 5: as big 1D array

```
        la $t9, arr # $t9 is base address of arr
        li $t0, 0   # $t0 is offset
        li $t2, 48  # $t2 is 6*8 (rows * columns)

loop:
        beq $t0, $t2, end # finish loop when index == 48

        # compute row number i to store: row = offset / 8
        # ( this works since j < 8 and we use integer
        #   division, and hence offset/8 == (offset-j)/8 )
        srl $t1, $t0, 3
        # store in arr + offset
        add $t4, $t9, $t0
        sb $t1, 0($t4)    # arr[i][j] = I

        # move to next iteration
        addi $t0, $t0, 1
        j loop
end:
```

Question 5: unroll inner loop

```
       la $t9, arr # $t9 is start address of row arr[i]
       li $t0, 0   # $t0 is i
       li $t2, 6   # $t2 is number of rows

outer: beq $t0, $t2, end_outer # finish when i == 6

       # unroll the inner loop (it has exactly 8 iterations)
       sb $t0, 0($t9)       # arr[i][0] = i
       sb $t0, 1($t9)       # arr[i][1] = i
       sb $t0, 2($t9)       # arr[i][2] = i
       sb $t0, 3($t9)       # arr[i][3] = i
       sb $t0, 4($t9)       # arr[i][4] = i
       sb $t0, 5($t9)       # arr[i][5] = i
       sb $t0, 6($t9)       # arr[i][6] = i
       sb $t0, 7($t9)       # arr[i][7] = i
       # move to next row
       addi $t9, $t9, 8
       addi $t0, $t0, 1
       j outer
end_outer:
```

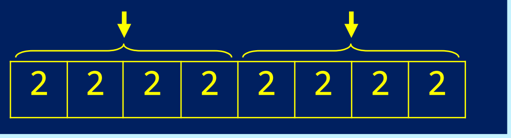


2 2 2 2 2 2 2 2

Question 5: unroll + combine

```
.align 2 # make sure the next label is word aligned!
arr:   .byte  0:48  # char array with 6 rows and 8 columns
```

```
       la $t9, arr # $t9 is start address of row arr[i]
       li $t0, 0   # $t0 is i
       li $t2, 6   # $t2 is number of rows
       li $t3, 0x00000000 # value added to $t3 after each row
       li $t4, 0x01010101 # add 1 to every 8-bit number in $t3
       # (only works because we have less than 256 rows!)
outer: beq $t0, $t2, end_outer # finish when i == 6
       # store 8 bytes using two word-stores
       sw $t3, 0($t9)     # handles columns 0-3 of row i
       sw $t3, 4($t9)     # handles columns 4-7 of row i
       # move to next row
       add $t3, $t3, $t4 # update row numbers being stored
       addi $t9, $t9, 8
       addi $t0, $t0, 1
       j outer
end_outer:
```



2 2 2 2 2 2 2 2