

## CS 61 - Programming Assignment 5

---

### Objective

The purpose of this final programming assignment is to test the limits of your LC3 skill by giving you a challenging implementation problem straight out of the textbook.

### High Level Description

There is a server somewhere, called the Busyness Server. This server tracks whether 16 different machines connected to it are **busy (0)** or **free (1)**: this information is stored in a single LC-3 word (aka a "bit-vector").

You will write a menu-driven system that allows the user to query this bit-vector as to the availability of the 16 machines, in various combinations.

### Before You Start Coding

This assignment is based on a variation of [Question 9.9](#) (p. 242 of the textbook) – which will require that you first work through the following:

- Example 2.11 (p. 37)
- Question 2.36 (p. 46)
- Question 5.6 (p. 146)

*(Note: I have lightly edited the wording of the following excerpts from the text for increased clarity)*

#### Example 2.11

Suppose we have eight machines that we want to monitor with respect to their availability. We can keep track of them with an 8-bit BUSYNESS vector, where a bit is 1 if the unit is free and 0 if the unit is busy. The bits are labeled 0 through 7, from right (lsb) to left (msb).

The BUSYNESS bit vector 11000010 corresponds to the situation where only units 7, 6, and 1 are free, and therefore available for work assignment.

Suppose work is assigned to unit 7. We update our BUSYNESS bit vector by performing the logical AND, where our two operands are the current bit vector 11000010 and the bit mask 01111111. The purpose of the bit mask is to clear (*set to 0*) bit 7 of the BUSYNESS bit vector, without affecting the remaining 7 bits. The result is the bit vector 01000010, which replaces the previous BUSYNESS.

Recall that we encountered the concept of bit mask in Example 2.7. Recall that a bit mask enables one to interact with some bits of a binary pattern while ignoring the rest. In this case, the bit mask clears bit 7 and leaves unchanged (ignores) bits 6 through 0.

Suppose unit 5 finishes its task and becomes idle. We can update the BUSYNESS bit vector by performing the logical OR of it with the bit mask 00100000. The result is 01100010, which again replaces the previous BUSYNESS.

### Question 2.36 (see Homework 2)

Refer to Example 2.11 (above) for the following questions.

- A. What mask value and what operation would one use to set machine 2 from free (1) to busy (0)? (*Without changing the state of the other machines, obviously!*)
- B. What mask value and what operation would one use to set machines 2 and 6 to free (1)? (*Again, without changing the state of the other machines.*)  
*Note: This can be done with a single boolean operation, but it cannot be implemented in LC-3 assembly language with a single instruction!*
- C. What mask value and what operation would one use to set **all** machines to busy (0)?
- D. What mask value and what operation would one use to set **all** machines to free (1)?
- E. Develop a procedure to "interrogate" the status of machine 2 by isolating it as the "sign bit" (i.e. the msb).  
For example, if the BUSYNESS pattern is 0101 1100, then the output of this procedure is 10000000. If the BUSYNESS pattern is 0111 0011, then the output is 0110 0000. In general, if the BUSYNESS pattern is:

b7	b6	b5	b4	b3	b2	b1	b0
----	----	----	----	----	----	----	----

the output is

b2	b1	b0	0	0	0	0	0
----	----	----	---	---	---	---	---

Hint: What happens when you ADD a bit pattern to itself?

### Question 5.6

Recall the machine busy example from Section 2.7.1. Assuming the BUSYNESS bit vector is stored in R2, we can use the LC3 instruction AND R3, R2, #1 to determine whether machine 0 is busy or not. If the result of this instruction is 0, then machine 0 is busy.

- A. Write an LC3 instruction that determines whether machine 2 is busy.
- B. Write an LC3 instruction that determines whether both machines 2 and 3 are busy.
- C. Write an LC3 instruction that indicates whether none of the machines are busy.
- D. Can you write a single LC3 instruction that determines whether machine 6 is busy? Is there a problem here?

The variation of Question 9.9 that you will implement in this assignment:

(remember - 0 = busy; 1 = free)

- A. Check if all machines are busy; return 1 if all are busy, 0 otherwise (i.e. if any are free).
- B. Check if all machines are free; return 1 if all are free, 0 otherwise (i.e. if any are busy).
- C. Check how many machines are busy; return the number of busy machines.
- D. Check how many machines are free; return the number of free machines.
- E. Check the status of a specific machine whose number is passed as an argument in R1; return 1 if that machine is free, 0 if it is busy.
- F. Return the number of the first (lowest numbered, or rightmost) machine that is free - i.e. a number between 0 and 15. **If no machine is free, return -1 (= xFFFF)**  
e.g. if the busyness vector is 0101 0100, then the first available machine is #2  
(Remember - the lsb is always considered bit #0)

## Your Tasks

The assignment can be broken down into the following tasks:

1. Your main code block should call a MENU subroutine, which prints out a fancy looking menu with numerical options, allows the user to input a choice, and returns in **R1** the user's selection {1, 2, 3, 4, 5, 6, 7} If the user inputs a non-existent option, output an error message and re-print the menu; repeat until a valid entry is obtained.

Here is the menu that you will be using for the assignment:

*(it is given to you in the starter code as a single long string located at memory address **x6000**)*

```
*****
* The Busyness Server *
*****
1. Check to see whether all machines are busy
2. Check to see whether all machines are free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available machine
7. Quit
```

2. Write the following corresponding subroutines:

- A. MENU
- B. ALL\_MACHINES\_BUSY
- C. ALL\_MACHINES\_FREE
- D. NUM\_BUSY\_MACHINES
- E. NUM\_FREE\_MACHINES
- F. MACHINE\_STATUS
- G. FIRST\_FREE

plus the two helper subroutines:

```
GET_MACHINE_NUM
PRINT_NUM
```

=====

## Subroutine Headers

The following headers have been provided to you in the template.

**NOTE:** R1 is always used to return a numeric or character value; R2 to return a flag value

```
;------
; Subroutine: MENU
; Inputs: None
; Postcondition: The subroutine has printed out a menu with numerical options, allowed the
;                user to select an option, and returned the selected option.
; Return Value (R1): The option selected: #1, #2, #3, #4, #5, #6 or #7
;                no other return value is valid.
;------
```

```

;-----
; Subroutine: ALL_MACHINES_BUSY (#1)
; Inputs: None
; Postcondition: The subroutine has returned a value indicating whether all machines are busy
; Return value (R2): 1 if all machines are busy, 0 otherwise
;-----

;-----
; Subroutine: ALL_MACHINES_FREE (#2)
; Inputs: None
; Postcondition: The subroutine has returned a value indicating whether all machines are free
; Return value (R2): 1 if all machines are free, 0 otherwise
;-----

;-----
; Subroutine: NUM_BUSY_MACHINES (#3)
; Inputs: None
; Postcondition: The subroutine has returned the number of busy machines.
; Return Value (R1): The number of machines that are busy (0)
;-----

;-----
; Subroutine: NUM_FREE_MACHINES (#4)
; Inputs: None
; Postcondition: The subroutine has returned the number of free machines
; Return Value (R1): The number of machines that are free (1)
;-----

;-----
; Subroutine: MACHINE_STATUS (#5)
; Input (R1): Which machine to check
; Postcondition: The subroutine has returned a value indicating whether the machine indicated
;                 by (R1) is busy or not.
; Return Value (R2): 0 if machine (R1) is busy, 1 if it is free
;-----

;-----
; Subroutine: FIRST_FREE (#6)
; Inputs: None
; Postcondition: The subroutine has returned a value in R1
;                 indicating the lowest numbered free machine
; Return Value (R1): the number of the free machine
;-----

```

### Helper subroutines (i.e. not included in menu)

Both subroutines are required for Option 5 (MACHINE\_STATUS);

PRINT\_NUM is required also for Options 3, 4 and 6

```
;-----  
; Subroutine: GET_MACHINE_NUM  
; Inputs: None  
; Postcondition: The number entered by the user at the keyboard has been converted into binary,  
;               and stored in R1. The number has been validated to be in the range {0,15}  
; Return Value (R1): The binary equivalent of the numeric keyboard entry  
; NOTE: You can use your code from assignment 4 for this subroutine, changing the prompt,  
;       and with the addition of validation to restrict acceptable values to the range {0,15}  
;-----
```

Your assignment 4 code is kind of overkill for this purpose, as it accepts & converts to binary all possible numbers in range {-32768, +32767}, while we only want {0, 15}

However, the code is already written & tested, so it is simpler to just package it as a subroutine and modify it a bit for the new purpose:

### Required changes to your assn 4 code:

- The binary value must be returned in **R1**, rather than the register specified for assn 4
- The prompt and error messages in assn 4 were provided as *remote* data, requiring the use of pointers to access them. You must substitute these with the new prompt & error messages, which are provided in the starter code as *local* data.

So in assn 4 you would probably have set up the string output like this:

```
LD R0, msg_pointer ; msg_pointer stores remote address of message string
```

In this subroutine version make sure you change that to:

```
LEA R0, msg_label ; msg_label is start of local message string
```

*If you have trouble understanding this advice, go back & redo labs 2 & 3!!)*

- Treat an initial '-' sign as an invalid input, exactly as you would a non-numeric input  
**NOTE:** This is different from the behavior depicted in sample screen 9 below, which shows user entry "-14": this spec requires that the '-' sign immediately trigger the error handler.
- After successfully converting the user input number, you must test that it is  $\leq 15$  (you already know that it is  $\geq 0$ , because you rejected negative inputs).  
If the the number is  $>15$ , it is invalid, so start over.

```
;-----  
; Subroutine: PRINT_NUM  
; Inputs: R1, guaranteed to be in range {0,15}  
; Postcondition: The subroutine has output the number in R1 as a decimal ascii string  
; Return Value : R1 (unchanged from input)  
;-----
```

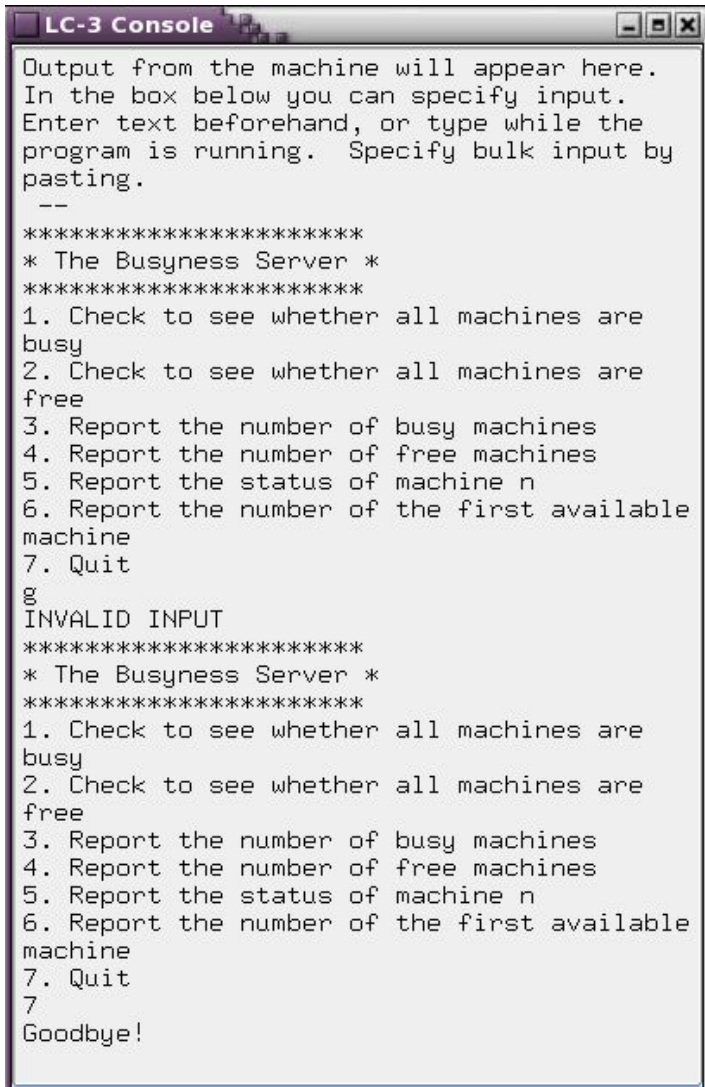
- Do **NOT** output a newline after the number output, as it may be part of a larger message.
- if  $n < 10$ , it is a single digit, and can be output as ascii by just adding x30
- if  $n \geq 10$ , it is a 2-digit number: the first digit is '1', and the second digit is (n-10)

## Expected/ Sample output

### Output

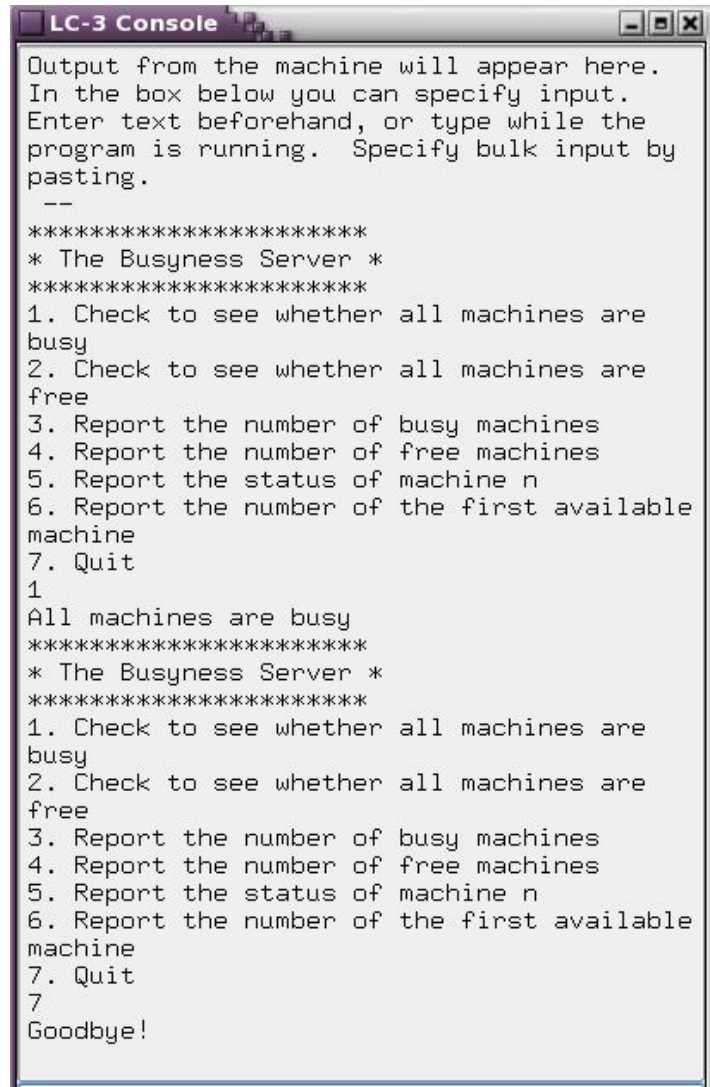
- The caption for each sample screen lists the corresponding menu selection & value of the test busyness vector
- Note the placing of newlines in the output!

### Examples



```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
g
INVALID INPUT
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

1. Wrong Input for menu, BUSYNESS = x0000



```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
1
All machines are busy
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

2. Option 1, BUSYNESS = x0000

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
1
Not all machines are busy
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

3. Option 1, BUSYNESS = xABCD

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
2
All machines are free
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

4. Option 2, BUSYNESS = xFFFF

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
2
Not all machines are free
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

5.(Option 2, BUSYNESS = xABCD

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
3
There are 6 busy machines
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

6. Option 3, BUSYNESS = xABCD



```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
4
There are 10 free machines
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

7. Option 4, BUSYNESS = xABCD

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
5
Enter which machine you want the status of
(0 - 15), followed by ENTER: 4
Machine 4 is busy
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

8. Option 5, BUSYNESS = x0000 with correct input

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
5
Enter which machine you want the status of
(0 - 15), followed by ENTER: -14
ERROR INVALID INPUT
Enter which machine you want the status of
(0 - 15), followed by ENTER: 9
Machine 9 is busy
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
```

9. Option 5, BUSYNESS = x0000 with incorrect input example #1

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
5
Enter which machine you want the status of
(0 - 15), followed by ENTER: q
ERROR INVALID INPUT
Enter which machine you want the status of
(0 - 15), followed by ENTER: 1
Machine 1 is busy
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
```

10. Option 5, BUSYNESS = x0000 with incorrect input example #2

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
6
No machines are free
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

11. Option 6, BUSYNESS = x0000

```
LC-3 Console
Output from the machine will appear here.
In the box below you can specify input.
Enter text beforehand, or type while the
program is running. Specify bulk input by
pasting.
--
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
6
The first available machine is number 0
*****
* The Busyness Server *
*****
1. Check to see whether all machines are
busy
2. Check to see whether all machines are
free
3. Report the number of busy machines
4. Report the number of free machines
5. Report the status of machine n
6. Report the number of the first available
machine
7. Quit
7
Goodbye!
```

12. Option 6, BUSYNESS = xABCD

#### Notes:

- As always, you must echo all inputs (no "ghost writing").
- When an invalid input is detected: Output the error message and re-prompt the user for input
- ***all outputs must be newline terminated***  
(except the number in machine status, which is internal to a larger message)

Your code will obviously be tested with a range of different busyness vectors: Make sure you do likewise!

#### Uh...help?

No individual component of this assignment is really difficult - but it has a lot of "moving parts" that all have to work together properly, so it can be very challenging, and take a long time to complete! So we'll give you some hints :)

## Really, Really Important

- **1 means free; 0 means busy**
- Store the BUSYNESS vector at the memory address **given to you in the template**.
  - The grader will test different values using this address
- Several of your subroutines require input validation: make sure that there is no way to exit such a subroutine except by providing valid input - in other words, if the user messes up, the subroutine must start over, with all prompts (*see e.g. the three Option 5 samples above*)

### Hint 01

All the subroutine headers have been provided in the starter code, along with all prompts and error messages. Use the headers as a guide to the construction of your subroutines.

In particular, detailed guides for the two helper functions are provided with their headers above.

**DO NOT ALTER ANY STRINGS OR ADDRESSES PROVIDED IN YOUR STARTER CODE!**

### Hint 02

**NEVER** backup (store/reload) the register used to return a value!

*In all subroutines, R1 is used to return a numeric or character value; R2 is used to return a flag value*

**ALWAYS** backup R0 and R7 if your subroutine invokes any Traps; plus all registers used as temporary storage in the course of the subroutine.

### Hint 03

Remember the algorithm you created to print out a number in binary: you examined each individual bit of a 16-bit register by "shifting" each bit into the msb in turn.

You will be using this same technique in several of your subroutines in this assignment.

### Hint 04 – sub A (MENU)

This subroutine gets a **single** character from the user. You do **NOT** have to press enter for the character to be accepted. The moment the user enters a character, it will be echoed, and the subroutine will try to validate the input {'1', '7'}; if valid, the option value will be returned in R1; if not, an error message should be output and the prompt repeated (i.e. start over).

Should you return the menu option as an ascii numeric digit, or as the corresponding number?

Either can be made to work, but which do you think is easier to test after return to main?

After responding to any menu selection other than #7, the menu must be re-presented, in an infinite loop (*see pseudo-code at end*).

The only way to exit the loop is by selecting menu option 7 ("Goodbye").

### Hint 05 – subs B & C (ALL BUSY/ALL FREE?)

If a machine is free, then it is NOT busy. If a machine is busy, then it is NOT free - i.e. you can use the exact same code structure for these two subroutines.

In both cases, return flag R2 = 1 for "affirmative" (all machines are busy/free respectively).

### Hint 06 – subs D & E (HOW MANY BUSY/FREE?)

These two subroutines form a pair like subs B & C above - you can use the same logic for both.



### Hint 07 – sub F (MACHINE STATUS)

The first step is to obtain user input by invoking the helper subroutine GET\_MACHINE\_NUM. This will return a validated number from 0 to 15 in R1, to be passed directly to subroutine F. The same number is also passed, still in R1, to the other helper subroutine PRINT\_NUM when printing the full status report.

The only acceptable return values from GET\_MACHINE\_NUM are in the range **{0, 15}**.

#### Examples of valid input:

*(sub returns with value in R1)*

- 12
- 0
- 0003
- +4

#### Examples of invalid input:

*(error message, repeat prompt)*

- -12
- 16
- +2g
- g

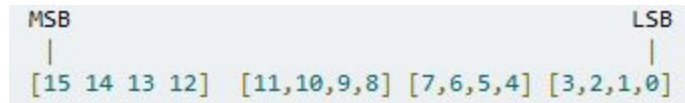
*See also the guide provided with the helper subroutine headers above.*

### Machine numbers for subroutine F:

The FIRST machine is machine 0 (*the lsb, or rightmost bit*);

the LAST machine is machine 15 (*the msb, or leftmost digit*)

Machine IDs:



Subroutine F can assume that the value in R1 has been properly validated {0, 15}

If machine (R1) is busy, return flag R2 = 0; if it is free, return R2 = 1

### Hint 08 – sub G (FIRST\_FREE)

As for sub F: the "lowest numbered" machine is the rightmost bit, aka the lsb, aka bit 0

### Hint 09

To implement your menu-driven system, construct an infinite loop that works like this:

*NOTE: Print (R1) means invoke PRINT\_NUM(R1)*

*If you needed to print the contents of a different register, you would have to R1 <= Rx first.*

```
while( true )
{
    R1 = menu()                ; call MENU subroutine

    if (R1 == 1)
        R2 = all_machines_busy()    ; call ALL_MACHINES_BUSY subroutine
        if (R2 == 1)
            Print "All machines are busy\n"
        else
            Print "Not all machines are busy\n"

    else if (choice == 2)
        [similar]

    else if (choice == 3)
        R1 = num_busy_machines()    ; call NUM_BUSY_MACHINES subroutine
        Print "There are ", (R1), " machines busy\n"

    else if (choice == 4)
        [similar]

    else if (choice == 5)
        R1 = get_machine_number()    ; validated user entry from #0 to #15
        R2 = machine_status(R1)      ; call MACHINE_STATUS and pass in R1
        if (R2 == 1)
            Print "Machine ", (R1), " is free\n"
        else
            Print "Machine ", (R1), " is busy\n"

    else if (choice == 6)
        R1 = first_machine_free()    ; call FIRST_FREE subroutine
        if (R1 == -1 )               ; #-1 == xFFFF
            Print "No machines are free"
        else
            Print "The first available machine is number ", (R1), '\n'
            ; don't forget to terminate this o/p with a nl!
            ; all other messages have their terminal nl built in.

    else if (choice == 7)
        Print "Goodbye!\n"
        HALT
}
```

## Submission Instructions

Submit ("Upload") your **assignment5.asm** file (*and ONLY that file!*) to the Programming Assignment 5 folder in Gradescope: the Autograder will run & report your grade within a minute or so.

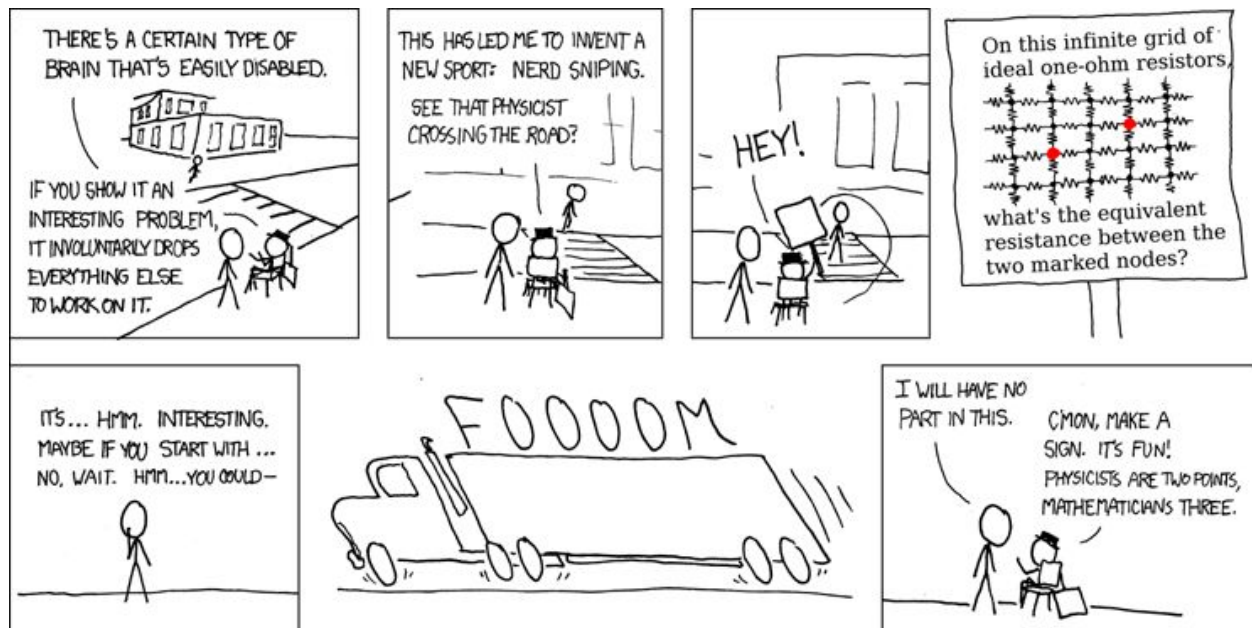
You may submit as many times as you like - your grade will be that of your last submission.

*If you wish to set your grade to a previous submission with a higher score, you may open your "Submission history" and "Activate" any other submission - that's the one we will see.*

## Rubric

- To pass the assignment, you need a score of  $\geq 80\%$ .  
The autograder will run several tests on your code, and assign a grade for each.  
But certain errors (*run-time errors, incorrect usage of I/O routines, missing newlines, etc.*) may cause ALL tests to fail  $\Rightarrow 0/100$ ! So submit early and study the autograder report carefully!!
- **You must use the template we provide** - if you make any changes to the provided starter code, the autograder may not be able to interpret the output, resulting in a grade of 0.

## One last XKCD comic for the quarter!



Source: <http://xkcd.com/356/>