HUMBER COLLEGE **Winter 2017**
COMPUTER SYSTEMS ARCHITECTURE **LAB 8,9**

CENG CPU-2017

The goal is to write a C program that will simulate a simple CPU. The simulation will be at a register level (i.e. will simulate register to register transfers). Your program should have a simple user interface that would allow it to be used to test machine level programs written for the CPU (read a binary file into memory, display memory, single step instructions, display registers, etc.) While single stepping the program should display the registers after the end of each instruction cycle.

Today's Assignment

The design of this CPU will be loosely based on the ARM Cortex M0+.

1. Continue implementing the execute stage of the CPU. In particular, implement:
    a. implement the Unconditional Branch instructions,
    b. implement the Conditional Branch instructions, and
    c. start implementing the Data Processing instructions in particular the ADD and SUB instructions.

    Refer to the CPU documentation for information on instruction coding.

    Approaches to dealing with coding will be discussed in class.

2. **Lab report**
    Hand in a copy of the code fragments that:
    a. implements determining the instruction type; and
    b. implements the immediate instructions including the sign, carry and zero flags.

    Include the code implementing the registers and any other code required to make your code understandable, e.g., defines used by your code.

CENG CPU-2017 Documentation Draft

Registers

R0-R15	- (16 32-bits) general purpose registers
	Special Uses: R13 – SP – Stack Pointer
	R14 – LR – Link Register
	R15 – PC – Program Counter
CCR	- Condition Codes - Sign, Zero and Carry flags
MBR	- (32-bits) Memory Buffer Register - buffer for all data
MAR	- (32-bits) Memory Address Register - Holds memory address
IR0, IR1	- (16-bits) 2 Instruction Registers
Stop flag	- flag set by STOP instruction
IR active flag	- selects active IR

Memory
Memory is a byte addressable array. Only 16K (0x4000) will be present for now.

Reset
On start up the CPU clears all of the registers and flags, and begins an instruction fetch.

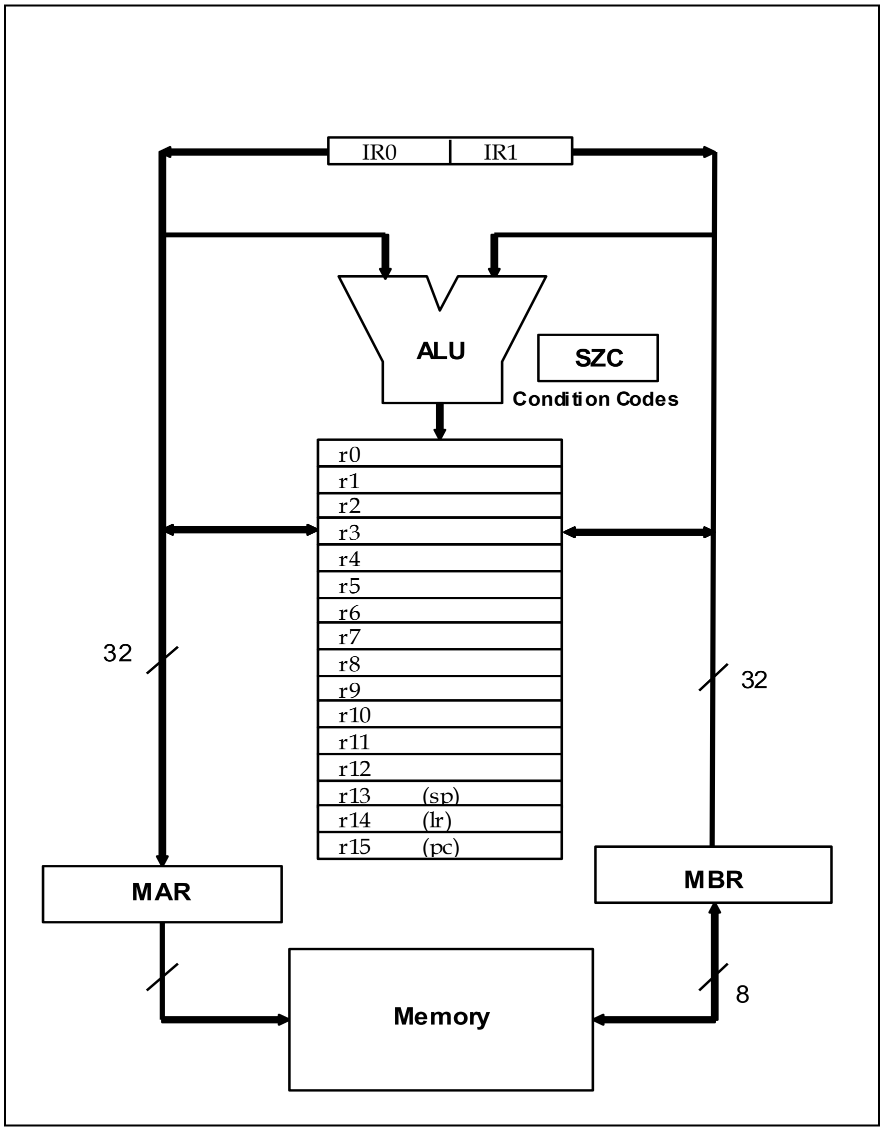


Figure - Virtual CPU-2017-Draft

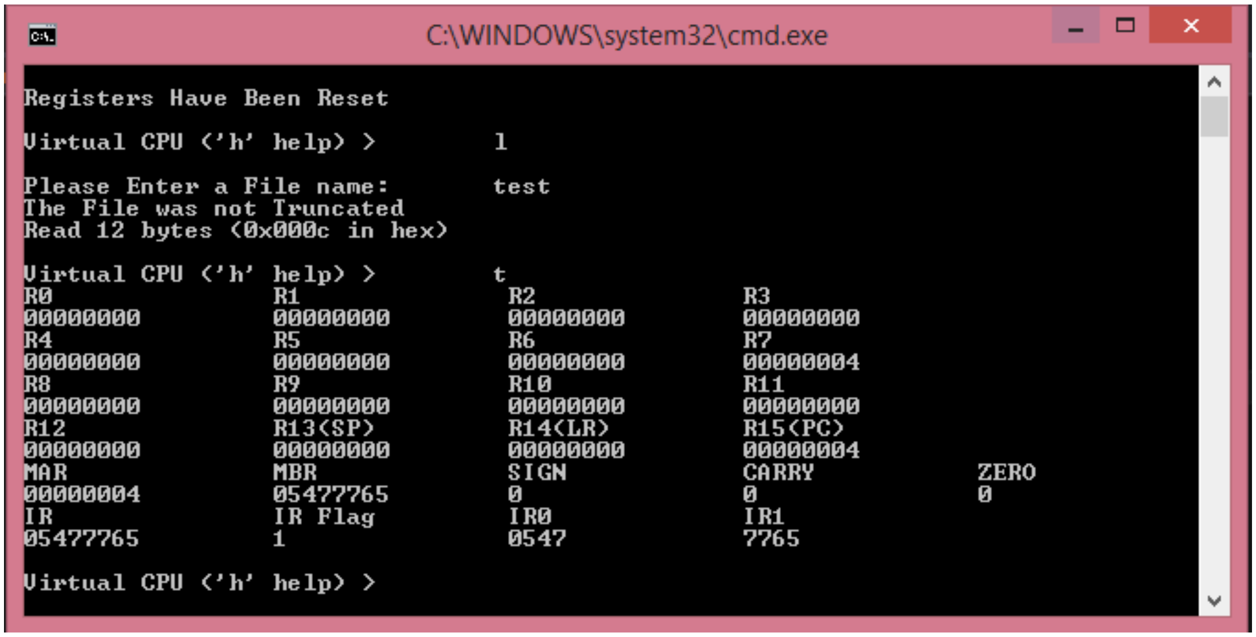
HINTS:
The fetch command is still responsible for receiving the current command from memory and pushing it into the IR and MBR. While the instruction cycle command is responsible for executing the correct instruction and performing fetch and execute. The execute function receives the current instruction and its type and then executes the instruction based on the given knowledge.
<u>Execute</u> – The execute part of the program is essential a giant if statement determining what to do with the given instruction. Essentially the instruction cycle function will pass a

command into the execute function and depending on the command it will modify the registers.

Add a function, which is responsible to run the instruction cycle until either stop flag is set or the program counter reaches the end of memory. All this function does is continue to execute the instruction in memory until the program counter reaches memory max or the stop flag is set.

Modify the flagsCheck function from previous lab that would be passing a variety of different registers into the function. It is capable of receiving the different registers and determining if the Sign, Carry or Zero flag is set.

Sample Output – the file test is just random data but as shown interpreting the byte data in the file and loading data into the registers.



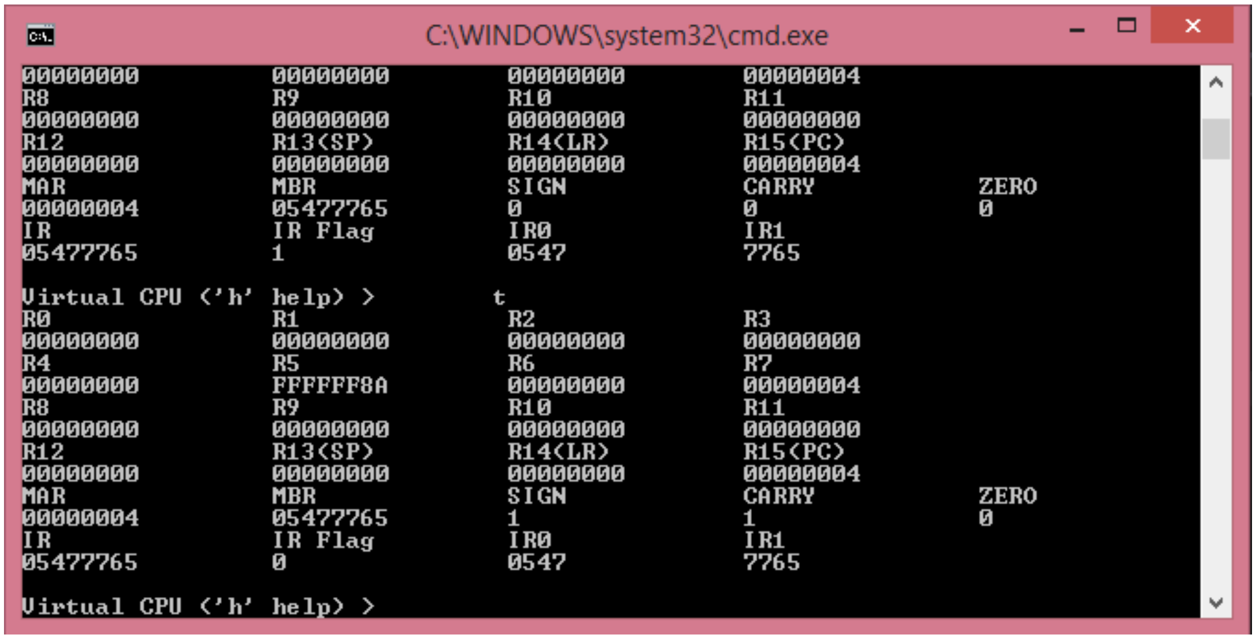
```
Registers Have Been Reset

Virtual CPU ('h' help) >        1

Please Enter a File name:       test
The File was not Truncated
Read 12 bytes (0x000c in hex)

Virtual CPU ('h' help) >        t
R0              R1              R2              R3
00000000        00000000        00000000        00000000
R4              R5              R6              R7
00000000        00000000        00000000        00000004
R8              R9              R10             R11
00000000        00000000        00000000        00000000
R12             R13(SP)         R14(LR)         R15(PC)
00000000        00000000        00000000        00000004
MAR             MBR             SIGN            CARRY           ZERO
00000004        05477765        0               0               0
IR              IR Flag         IR0             IR1
05477765        1               0547            7765

Virtual CPU ('h' help) >
```



```
00000000         00000000         00000000         00000004
R8               R9               R10              R11
00000000         00000000         00000000         00000000
R12              R13(SP)          R14(LR)          R15(PC)
00000000         00000000         00000000         00000004
MAR              MBR              SIGN             CARRY            ZERO
00000004         05477765         0                0                0
IR               IR Flag          IR0              IR1
05477765         1                0547             7765

Virtual CPU ('h' help) >         t
R0               R1               R2               R3
00000000         00000000         00000000         00000000
R4               R5               R6               R7
00000000         FFFFFF8A         00000000         00000004
R8               R9               R10              R11
00000000         00000000         00000000         00000000
R12              R13(SP)          R14(LR)          R15(PC)
00000000         00000000         00000000         00000004
MAR              MBR              SIGN             CARRY            ZERO
00000004         05477765         1                1                0
IR               IR Flag          IR0              IR1
05477765         0                0547             7765

Virtual CPU ('h' help) >
```

Sample Code:

```
#define MEMORY_MAX 0x3E80   //Determines maximum memory size (16000)
#define MAX32 0xFFFFFFFF
#define HEX_COLUMN 16
#define REG_NUM 16                 //Determines amount of registers

/* Defines all registers and flags */
#define SP R[13]                   //Stack Pointer
#define LR R[14]                   //Link Register
#define PC R[15]                   //Program Counter
#define IR IR                      //Instruction Register Combination of IR0 & IR1
#define ALU ALU                    //Arithmetic Logic Unit
#define MAR MAR                    //Memory Address Register
#define MBR MBR                    //Memory Buffer Register
#define IR0 IR0                    //Instruction Register 0
#define IR1 IR1                    //Instruction Register 1
#define SIGN SIGN                  //Sign Flag
#define CARRY CARRY                //Carry Flag
#define ZERO ZERO                  //Zero Flag
#define STOP_FLAG STOP_FLAG //Flag Set by Stop Instruction
#define IR_FLAG IR_FLAG            //Flag to determine the active Instruction Register
```