**Dalhousie University Faculty of Engineering**

**ECED 3403**

Assignment 2: Emulator

Testing

Submitted by:

Ray Su B00634512

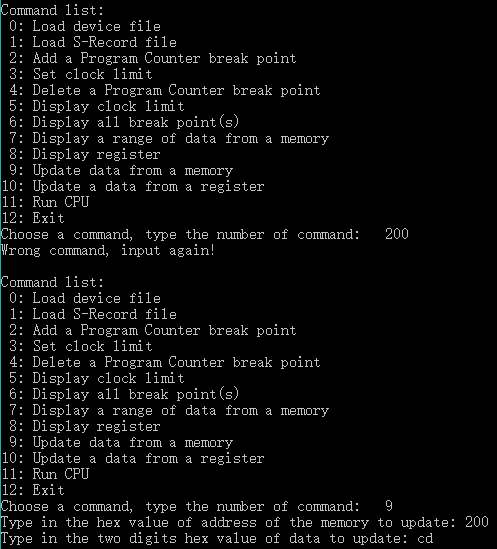
* Debugger test:
* Test to set and display data in specific memory

Expect test result:

Choose command 9 and input data value to set. And then choose command 7 and the value displayed is as same as the value inputted.

Actual test:

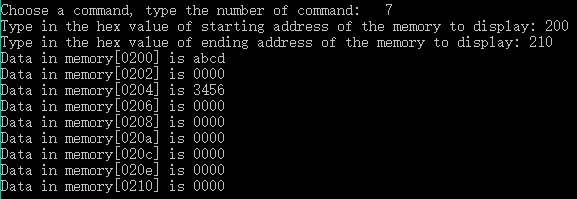
Input memory address(200) and memory value(#$ABCD):



Input memory address(204) and memory value(#$3456):



Display data stored from mem[200] to mem[210]:



Test result:

Data can be set and displayed successfully through debugger.

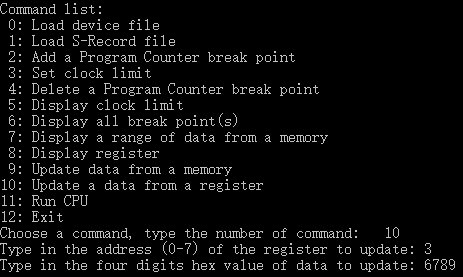
* Test to set and display data in specific register

Expect test result:

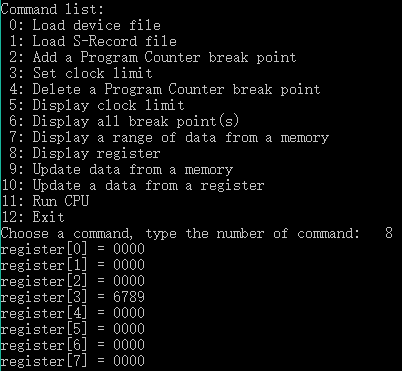
Choose command and input data value to set. And then choose command and the value displayed is as same as the value inputted.

Actual test:

Input register address(3) and value(#$6789):



Display data stored in that register[3]:



Test result:

Data can be store and display from register successfully through debugger.

* Test of loader:

Test Input:

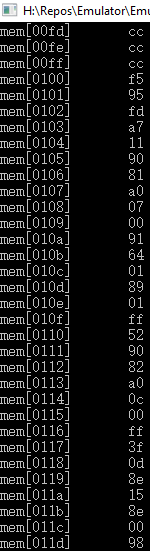
S00E0000Strings.asm98

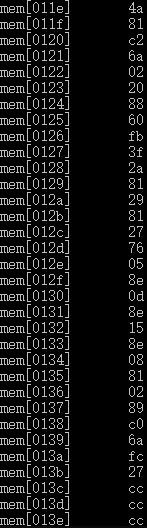
S1210100F595FDA7119081A007009164018901FF529082A00C00FF3F0D8E158E009843

S121011E4A81C26A02208860FB3F2A8129812776058E0D8E158E08810289C06AFC2760

S9030100FB

Actual test: memory print to console:







Expect output:

Origin is #$0100.

For the first S1 record, starting from address 100, the first byte of data should be F5. For the second S1 record, starting from address 11E, the first byte of data should be 4A. The final Program Counter(R7) should point to 100.

Test result:

The actual output matches expect output. The PC is 100 by showing R7 value. So loader is functional.

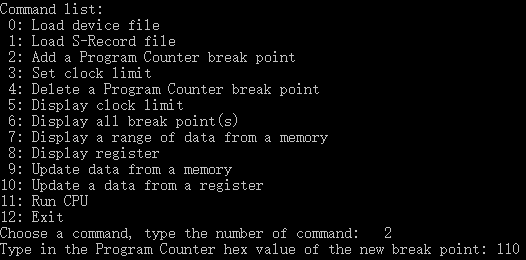
* Debugger program counter break point test:

Expect test result:

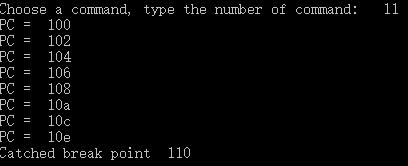
Set break point at PC = 110. Then CPU will be stop at PC = 110 and user will have access to debugger.

Actual test:

Set initial PC(R7) value to 100:

Set a break point at 110:

Run CPU and print out current program counter:



Test result:

CPU was stop at PC = 110 and gave the access to debugger. So break point function is working.

* Clock limit test:

Expect result:

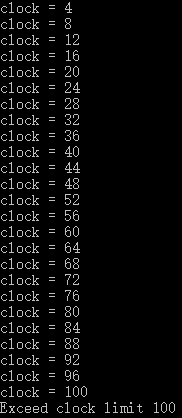
Set clock limit at 100 and CPU will be stop at clock = 100.

Actual test:

Set clock limit to 100:



Run CPU and display clock at each cycle:



Test result:

CPU stopped when reached clock limit so clock limit is working.

* Fetch test:

Expect test result:

Load instruction from memory to Instruction Register, and program counter increased by two. The example instruction is BAL: 3FFF (

Actual test:

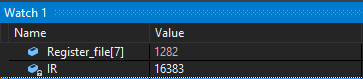
Load instruction to memory [500] through debugger:



Set program counter(R7) to 500:



Set a break point at the end of fetch and add PC and IR to watch, run CPU:



PC (R7) = 1282 = #$502

IR = 16383 = 0b 0011 1111 1111 1111 (BAL)

Test result:

PC increased by two and instruction was loaded to IR, so fetch is functional.

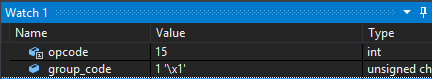
* Decode test:

Expect test result:

Recognize the first three bits of instruction stored in IR and get the Opcode ready for execution. Follow the fetch test, the first three digits (group\_code) should be 001 (1). Opcode should be 15 (0b001111).

Actual test:

Set a break point at the end of decode and add Opcode and the first three digits (group\_code) to watch:



Test result:

Opcode is set correctly and ready for execution. Decode is working as expect.

* Execution test: (Execute tests are based on pervious test result, fetch and decode succeed)
* Loading instruction test: LD R1-,R0

R1 = 100, mem[100] = 1234. (Instruction stored in mem[500])

S-Record: S105050008826B,S9030500F7, Origin is #$0500

Expect test result:

Data in mem[100] (1234) will be load to R0, R1 will decrease by 2 after loading

Actual test:

Set mem[100] = 1234:





Set R1 = 100:



Load S-Record and run CPU, display R0:



Display R1:



Test result:

Data in R0 is #$1234, LD function execution is correct.

* Storing instruction test: str R1,R0,4

R1 = #$aaaa, R0 = 100. (Instruction stored in mem[500])

S-Record: S105050008E20B,S9030500F7, Origin is #$0500

Expect test result:

Data in R1 (#$aaaa) will be load to mem[104]

Actual test:

Set R0 to 100:



Set R1 to #$aaaa:



Run S-Record and display mem[104]:



Test result:

Data (#$AAAA) has been loaded to mem[104], so STR is functional.

* Register initialization instruction test: movh #$6677,R1

R1 was set to #$1111

S-Record: S105050031A321,S9030500F7, Origin is #$0500

Expect test result:

High byte of data #$6666 will be load to high byte of R1

Actual test:

Set R1 to #$1111:



Run S-Record and display R1:



Test result:

High byte of data has been load to high byte of R1, so R1 = #$6611

* Branching instruction test: blt #$600

S-Record: S10505007F383E,S9030500F7, Origin is #$0500

Expect test result:

PSW.N XOR PSW.V = 1, is PC to 600

Actual test:

Set PSW.N:



Run S-Record and display PC (R7):



Test result:

PC was updated to 600 when PSW.N set and PSW.V cleared. BLT is functional.

* ADDC testing: addc R1,R2

R1 = 1111, R2 = 89ab, PSW.C = 1

S-Record: S10505000A6289, S9030500F7

Expect test result:

1111+89ab+1= 9abd, and store the result to R2. PSW.C, PSW.Z, PSW.N and PSW.V will be cleared.

Actual test:

Set R1 to 1111:



Set R2 to 89ab:



Set PSW.C = 1:



Run S-Record, display R2:



Display PSW:



Test result:

Result 9abd is correct and stored in R2, PSW.C, PSW.Z, PSW.N and PSW.V were cleared. So ADDC is working.

* CMP test: CMP R1,R0

R0=#$aaaa, R1 = #$aaaa

S1050500086A83, S9030500F7

Expect test result:

After comparing two data in register and update PSW.Z. Compare #$aaaa(R1) and #$aaaa(R0), PSW.Z = 1.

Actual test:

Set R0 to #$aaaa:



Set R1 to #$aaaa:



Run S-Record, display PSW(R6):



Test result:

PSW.Z has been set, CMP is working.

* DADD testing: DADD R1,R0

R0=#$6789, R1=#$5432

S1050500086885, S9030500F7

Expect test result:

6789+5432 = 12221, so R0 = #$2221, PSW.C = 1

Actual test:

Set R0 to #$6789:



Set R1 to #$5432:



Run S-Record and display R0:



Display PSW:



Test result:

Result 2221 stored in R0 and PSW.C is set, so DADD is working.

* RRC test: RRC R0

R0 = #$aaaa = 0b 1010 1010 1010 1010, PSW.C = 1

S1050500007382, S9030500F7

Expect test result:

After rotation, R0 = #$d555 = 0b 1101 0101 0101 0101, PSW.C=0

Actual test:

Set R0 to #$aaaa:



Set PSW.C:



Run S-Record and display R0:



Display PSW:



Test result:

R0 equal to #$d555 and PSW.C = 0. So RRC is working well.

* SWPB test: SWPB R0

R0 = #$1234

S1050500007580, S9030500F7

Expect result:

After switch, R0 = #$3412

Actual test:

Set R0 to #$1234:



Run S-Record and display R0:



Test result:

High byte and low byte in R0 has been switched. So SWPB is working well.

* SXT test: SXT R0

R0 = #$00F0

S1050500007580, S9030500F7

Expect result:

After sign extension, R0 should be equal to #$FFF0

Actual test:

Set R0 to #$00F0



Run S-Record and display R0:



Test result:

Sign extension has been set.

* Device and Interrupt testing:

Set device 0 to 3 as input device, device 4 to 7 as output device. Device 0 interrupt enabled. Only device 0 and device will be used in this testing. The ISR of device 0 will store the input data to device 4’s CSR.DATA. Device 4 will output data to an output file after processing time. Before run CPU, debugger will load S-Record to Memory, which include ISR and device vector’s setting. Debugger will load device file, which include device information and input data schedule.

Expect test result:

Input data will be output to output file after a specific time.

Input file:

ASM file:

dev0 equ #$0000

dev0\_psw equ #$0060 ;priority of device 0 is 3, IE is set

dev0\_vec equ #$FFC0

isr0\_addr equ #$FFC2 ;address of ISR0

dev4 equ #$0008

dev4\_psw equ #$0080 ;priority of device 1 is 4, IE is unset

dev4\_vec equ #$FFD0

isr4\_addr equ #$FFD2 ;address of ISR2

comp\_psw equ #$0020 ;priority of computer is 1

PSW equ R6 ;PSW is alias of R6

LR equ R4 ; LR is alias of R4

SP equ R5 ; SP is alias of R5

PC equ R7 ; PC is alias of R7

STKTOP equ #$FFBE ; Top-of-stack - word below device vectors

org #$500

Start movl comp\_psw,PSW ;set computer priority equal to 1

; Initialize stack point

movl STKTOP,SP ; LSB to SP

movh STKTOP,SP ; MSB to SP

;turn on the PSW.IE of device 1

movl dev0,R0

movh dev0,R0

ld R0,R1

add #1,R1

st R1,R0

wait

bal wait

org #$2000

isr0

movl dev0,R0 ;set device 1 address

movh dev0,R0

;movl dev4,R1 ;set device 4 address

;movh dev4,R1

ldr.b R0,#1,R2

;ld.b R1,R2

str.b R2,R0,#9 ;load data in dev0 to dev 4

mov LR,PC

;

org #$ffc0

;set device 0 priority to 3

word #$0060

word isr0

;set device 4 priority to 4

org #$ffd0

word #$80

end Start

Translated S-Record:

S0100000interrupt.asm93

S11505000691F595FDA7009000A0018089600888FF3FB8

S10D2000009000A0C2C0D0E42776CF

S107FFC060000020B9

S105FFD08000AB

S9030500F7

Device input file:

0 1 0

1 1 0

2 1 0

3 1 0

4 0 35

5 0 40

6 0 40

7 0 50

150 0 C

250 0 2

350 0 A

450 0 D

Load file and run CPU:



Set clock limit to 1000:



Run CPU until reach clock limit:



Device Output file:

at time 225, device 4 output C

at time 327, device 4 output 2

at time 425, device 4 output A

at time 527, device 4 output D

Test result:

Data output is correct. Device and interrupt is functional.

**Dalhousie University Faculty of Engineering**

**ECED 3403**

Assignment 2: Emulator

Code

Submitted by:

Ray Su B00634512