ECE586 FINAL PROJECT
PDP-11 Instruction Set Architecture Simulator

Project Members:
Shadman Shamin
Chandan Muralidhar
Sourabh Balakrishna
Apurva Gandole

Design Specification:

The objective of the project is to design a simulator for PDP-11 Instruction Set Architecture. The design is simulating the functionality of PDP-11 architecture. We have deigned the simulator in high level language i.e. C language which mimics the microprocessor by reading instructions and maintaining internal variables which represent the processor's register.

The block diagram representation of the PDP-11 ISA Simulator implementation is given below,

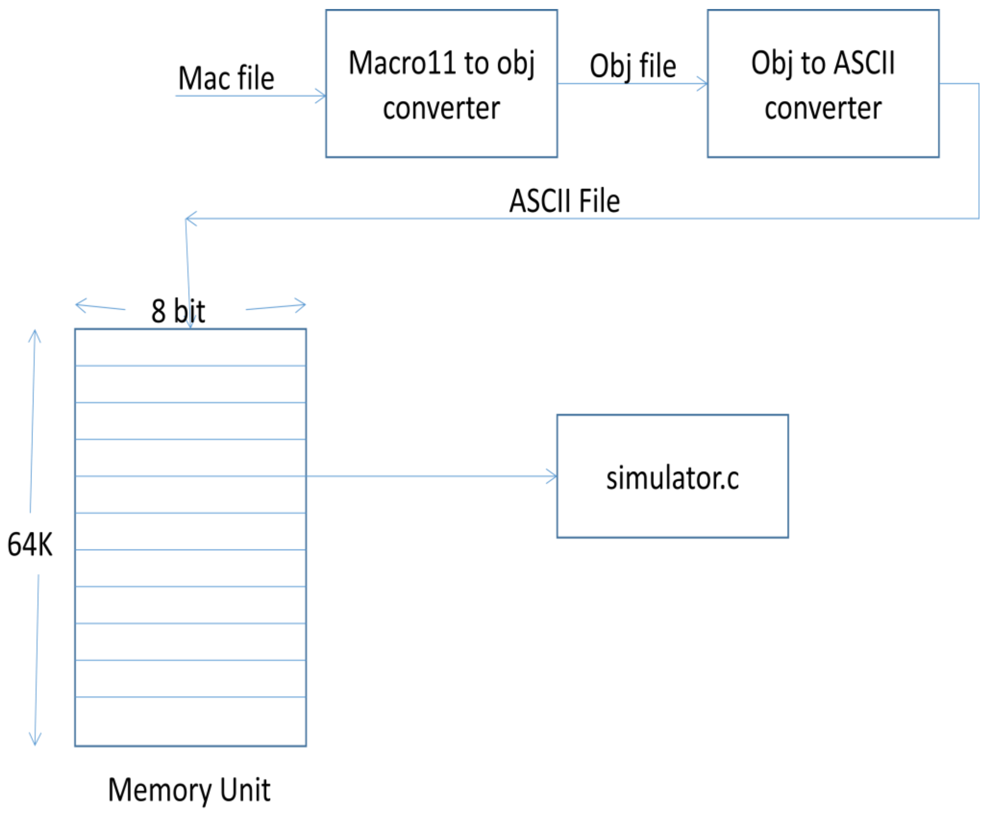

Figure1. Block Diagram for PDP-11 implementation

Theory of Operation:

Theory of Operation:

The makefile provides recipes to allow the user to run different type commands. Here are a list of the recipes provided inside the makefile.

- **all** - run a .mac files through the simulator
- **All** - run a .ascii file through the simulator

- **macro11** - converts mac file to an object file
- **obj2ascii** - converts object file to an ascii file
- **compile** - compiles the header and C files
- **run** - Runs the simulation environment
- **show** - displays the branch and trace files side by side
- **trace** - Just displays the trace file
- **branch** - Just displays the branch trace file
- **clean** - cleans temporary files before generating new files

Flags used :

- **DEBUG** - allow to print debug statements
- **STEP** - allows the user to do a step by step debugging
- **A** - enables memory alignment error
- **PRINTREG** - Prints Register and Flag contents for each Instruction

Variables used:

- **TEST** - Input the filename either mac or ascii. The files should be put inside the folder Testfiles
- **CFLAGS** - Input the flags needed.

Execution Steps:

1. // Change the working directory to code
   $ cd Code

2. // If the user just wants to run all the files without debugging and load a mac file
   $ make all TEST=filename.mac CFLAGS=”-D A”

   OR

   // If the user just wants to run all the file with debugging and load a mac file
   $ make all TEST=filename.mac CFLAGS=”-D A -D STEP -D DEBUG -D PRINTREG”

   OR

   // If the user just wants to run all the file with debugging and load an ascii file
   $ make ALL TEST=filename.ascii CFLAGS=”-D A -D STEP -D DEBUG -D PRINTREG”

3. // Display trace file side by side
   make show

Test Plan:

All Register Addressing Modes: We have tested all the addressing modes for PDP-11 architecture. We have verified the result with the generated ASCII file.

; Register Addressing Modes Test

```
START:MOV #1, @#600          ; Pushing 1 into mem[100]
      MOV #2, @#602          ; Pushing 2 into mem[102]
      MOV #3, @#604              ; Pushing 3 into mem[104]
      MOV #4, @#606              ; Pushing 4 into mem[106]
      MOV #600, @#700            ; Pushing 100 into mem[200]
      MOV #602, @#702            ; Pushing 102 into mem[202]
      MOV #604, @#704            ; Pushing 104 into mem[204]
      MOV #606, @#706            ; Pushing 106 into mem[206]
      MOV #70, R0                ; R0 -> 70
      MOV R0, R1                 ; R1 -> 70
      MOV #25, (R1)              ; Move 25 into the mem[Reg[R1]]: MEM[70] -> 25
      ADD (R0), (R1)             ; Register Deferred for both destination and source -
      contents of location mem[Reg[R1]] + me[Reg[R2]]: MEM[70] -> 50
      MOV (R1), R0               ; Register Deferred: R0 -> 50
      MOV (R1), (R0)+            ; Auto Increment: MEM[50] -> 50
      MOV #51,(R0)+              ; Auto Increment: MEM[52] -> 51
      MOV #52, (R0)+             ; Auto Increment: MEM[54] -> 52
      MOV -(R0), (R0)+           ; Auto Decrement: MEM[52] -> 52
      MOV @#52, R1               ; Auto Decrement: R1 -> 52
      MOV #700, R0               ; R0 -> 200
      MOV @(R0)+, R1             ; R1 -> 1      -- Auto Increment Deferred
      MOV @(R0)+, R2             ; R2 -> 2      -- Auto Increment Deferred
      MOV @(R0)+, R3             ; R3 -> 3      -- Auto Increment Deferred
      MOV @(R0)+, R4             ; R4 -> 4      -- Auto Increment Deferred
      MOV @-(R0), R1             ; R1 -> 4      -- Auto Decrement Deferred
      MOV @-(R0), R2             ; R2 -> 3      -- Auto Decrement Deferred
      MOV @-(R0), R3             ; R3 -> 2      -- Auto Decrement Deferred
      CLR R4                     ; Clear R4
      MOV #40, 4(R0)             ; MEM[706] -> 40 - Index
      MOV (R0),R4                ; R4 -> 40
      MOV #700, R5               ; R5-> 700
      MOV @4(R5), R1             ; Index Deferred: R1 -> 3
      HALT                       ; End of the program
```

Single Operand Testing: We have done the single operand testing for all instructions using simple register addressing mode. We have verified the result with the generated ASCII file.

```
; Single Operand Test


START:      MOV   #1, R1        ; R1->1
            CLR   R1            ; R1->0
            MOV   #2, R1        ; R2->2
            INC   R1            ; R1->R1+1
            MOV   #3, R1        ; R3->3
            DEC   R1            ; R1->R1-1
```

```
MOV   #4, R1           ; R1->4
ADC   R1               ; R1->R1+ Carry
MOV   #5, R1           ; R1->5
SBC   R1               ; R1-> R1- Carry
CLC                    ; Clear Carry
MOV   #7, R1           ; R1->7
TST   R1               ; load R1 (Flag Status: Z=0,N=0,C=0,V=0)
MOV   #100000, R1      ; R1->100000
NEG   R1               ; R1->R1-1
MOV   #10, R1          ; R1->10
COM   R1               ; R1-> ~R1
MOV   #2, R1           ; R1->2
ROR   R1               ; Rotate right 1 bit
MOV   #2, R1           ; R1->2
ROL   R1               ; Rotate left 1 bit
MOV   #2, R1           ; R1->2
ASR   R1               ; R1->R1>>1
MOV   #2, R1           ; R1->2
ASL   R1               ; R1->R1<<1
MOV   #177400, R1      ; R1->177400
SWAB R1                ; Rotate 8 bits
HALT                   ; End of the program
```

Double Operand Testing: We have done the double operand testing of all the instructions using register addressing mode. We have verified the result with the generated ASCII file.

```
; Double Operand Test

START:      MOV #5,R0    ;  R0->5
            MOV #2,R1    ;  R1->2
            ADD R0,R1    ;  R1-> R1+R0
            MOV #6,R1    ;  R1->6
            MOV #6,R2    ;  R2->6
            SUB R0,R1    ;  R1->R1-R0
            MOV #3,R1    ;  R1->3
            CMP R1,#3    ;  R1-3 (Flag Status: Z=1,N=0,C=1,V=0)
            MOV #2,R1    ;  R1->2
            MOV #1,R2    ;  R2->1
            BIT R1,R2    ;  R2&R1(Flag Status: Z=1,N=0,C=1,V=0)
            MOV #3,R1    ;  R1->3
            MOV #2,R2    ;  R2->2
            BIC R1,R2    ;  R2->R2&~R1(Flag Status: Z=1,N=0,C=1,V=0)
            MOV #5,R1    ;  R1->5
            MOV #1,R2    ;  R2->1
            BIS R1,R2    ;  R2->R2|R1(Flag Status: Z=0,N=1,C=1,V=0)
```

```
HALT                        ;  End of the program
```

Branch Instructions Testing: We have done the branch instruction testing of all branch instructions by setting and resetting the status flags. We have verified the result with the generated ASCII file.

```
BR L1                       ; Branch to L1

L2:     SEZ                 ; Z=1
        BEQ L3              ; If Z=1 branch to L3
L4:     CLV                 ; V=0
        BLT L5              ; Branch if less than: If N and V are different branch to L5
L6:     CCC
        SEN
        CLV
        CLZ
        BLE L7              ; Branch if less than or equal to:
L8:     MOV #-3, R0         ; R0->-3
        BMI L9              ; Branch if Minus: If negative is set we branch to L9
L10:    CCC
        SEC
        CLZ
        BLOS L11            ; Branch on lower than or same as: If zero or carry is 1 we branch
L12:    SEV
        BVS L13             ; Branch on overflow set: if overflow is set we branch
L14:    BCC L15
L16:    BLO END
L1:     CLZ                 ; Z=0
        BNE L2              ; If Z=0 branch to L2
L3:     SEN                 ; Set Negative
        SEV                 ; Set Overflow
        BGE L4              ; If they are both the same branch to L4
L5:     CCC
        CLN                 ; Set Negative to 1
        CLV                 ; Set Overflow to 1
        CLZ                 ; Clear Zero to 0
        BGT L6              ; Branch if greater than 0:
L7:     CCC
        MOV #5, R0
        BPL L8              ; Branch on Plus: If positive branch to L8
L9:     CLZ
        CLC
        BHI L10             ; Branch on higher than: if both Carry and Zero is 0 we branch
L11:    CLV
        BVC L12             ; Branch on overflow clear: if overflow is cleared we branch
```

```
L13:    CCC
        BCC L14             ; Branch if carry clear:
L15:    MOV #65535,R0
        MOV #1,R1
        ADD R0,R1
        BCS L16             ; Branch if carry is set:
END:    HALT                ; End of the program
```

Jump, JSR, RTS Instruction Testing: We have done the testing of JMP, JSR and RTS testing on PC. We have verified the result with the generated ASCII.

```
; JMP JSR RTS Instructions

START:      MOV #3, R0   ; R0 -> 3
            JSR PC, L2   ; Call to Subroutine L2
            MOV #1, R1   ; R1-> 1
            JMP LOCAT    ; Unconditional jump to LOCAT

 L2:        MOV #1, R2   ; R2 -> 1
            RTS PC       ; Return from Subroutine (PC should point to MOV instruction)

 LOCAT:     HALT         ; End of the program
```

BIT test cases: We have done the testing of BIT. We have verified the result by setting and resetting the condition code.

```
; BIT compare corner

START: MOV #1, R1        ; R1->1
       MOV #1, R2        ; R2->1
       BIT R1,R2         ; Z=1
       BEQ J1            ; Z=1, so jump to J1
       MOV #4, R4        ; R4->4
 J1:   SCC               ; set all flags for check
       CCC               ; all flags reset
       HALT              ; End of the program
```

Stack Addressing mode test cases: We have tested Deferred, Autoincrement, Autoincrement deferred, Autodecrement, Indexed, Indexed deferred addressing modes with stack pointer.

```
STACK: .BLKW 200
TOS:
MOV #TOS, SP
MOV #10,  (SP)
MOV #20, -(SP)  ; AUTODECREMENT
MOV #30, -(SP)
MOV #40, -(SP)
MOV (SP)+, R1   ; R1 -> 40; AUTOINCREMENT
MOV (SP)+, R2   ; R2 -> 30
MOV (SP)+, R3   ; R3 -> 20
MOV (SP),  R4    ;  R4 -> 10
MOV #50,  -(SP)
MOV #60,  -(SP)
MOV 2(SP), R5    ; R5 -> 50; INDEXED
MOV #5,   @#300  ;   STORE 5 IN LOCATION 300
MOV #300, -(SP) ;     STORE POINTER TO OPERAND 5 ON STACK
MOV @(SP)+, R1   ;  R1 -> 5; AUTOINCREMENT DEFERRED
MOV #5,   @#400  ;   STORE 5 IN LOCATION 400
MOV #400,  -(SP) ;    STORE POINTER TO OPERAND 5 ON STACK
MOV #10,   -(SP) ;
MOV #20,   -(SP) ;
MOV @4(SP), R1   ;  R1 -> 5; INDEX DEFERRED
HALT
```

Program Counter addressing modes test: We have tested Immediate, Absolute, Relative and Relative deferred addressing modes.

```
RELATIVE: .WORD 6.
START:        MOV #5, R1                 ; R1 ->5   IMMEDIATE MODE
              BR IMMD
IMMD:         CLR R1                     ; R1 ->0
              MOV #10, @#100             ; Absolute Addressing mode; MEM[100] -> 10
              MOV @#100, R1              ; R1 -> 10
              BR IMDR
IMDR:         CLR R1                     ; R1 -> 0
              MOV RELATIVE, R1           ; Relative address   R1->6
              BR RLDF
RLDF:         CLR R1                     ; R1->0
              MOV #10, @#6               ; Relative Deferred ; MEM[6]->10
              MOV @RELATIVE, R1          ; R1 -> 10
              HALT
```

References:

- ECE 586 Slides by Prof. Mark G. Faust.
- PDP-11 Handbook http://gordonbell.azurewebsites.net/digital/pdp%2011%20handbook%201969.pdf.
- PDP-11 Instruction reference by University of Toronto http://www.teach.cs.toronto.edu/~ajr/258/pdp11.pdf

- Wikipedia(https://en.wikipedia.org/wiki/PDP-11_architecture)