Assignment 5

CSCI131

Name: James Marino

Username: jm617

Student Number: 4720994

Table Of Contents

1. Executive Summary

This report aims to show compilation of BASIC code to PDP-11 Assembly to be run on the PDP-11 Machine. It uses Bison and Flex to look for statements and expressions in the BASIC code then appropriately insert C printf statements which in turn, when this C program is run it provides an output in PDP-11 assembly for each part of the BASIC program.

For example a MSG statement would know to insert a ‘writeline’ command in the final output. Please see code and results below.

2. Body

2.1 Flex Code

Flex program:

- Identifiers <=, >, ==, != were added to return GE, GT, EQ, NEQ
- IF, THEN, ENDIF statements added

scanner.lex

```
%option noyywrap

%{
	#include "parser.h"
	#include <stdio.h>
	#include <string.h>
%}


WHITE [ \t]+
NAME [a-z]+
INTNUM [0-9]+
QUOTESTRING \"(\\.|[^"])*\"

%%


PROGRAM 		return PROGRAM;
END 			return END;
VAR 			return DECLARE;
, 			return COMMA;
; 			return SEMICOLON;
"+" 			return PLUS;
"-" 			return MINUS;
"*" 			return MULTIPLY;
"/" 			return DIVIDE;
"(" 			return LPAR;
")" 			return RPAR;
"=" 			return ASSIGN;
"<" 			return LESS;
">=" 			return GE;

"<=" 			return LE;
">" 			return GT;
"==" 			return EQ;
"!=" 			return NEQ;

WHILE 			return WHILE;
DO 			return DO;
ENDWHILE 		return ENDWHILE;

IF 			return IF;
```

```
THEN                return THEN;
ENDIF               return ENDIF;

{QUOTESTRING}       {
       yylval.str_val = strdup(yytext);
       return QSTRING;
}
READ                return READ;
READOCT             return READOCT;
PRINT               return PRINT;
PRINTOCT            return PRINTOCT;
MSG                 return MSG;
NEWL                return NEWL;
{NAME}              { yylval.str_val = strdup(yytext);  return
IDENTIFIER; }
{INTNUM}           { yylval.ival = atoi(yytext); return NUMBER; }
{WHITE}             ;
\n                  { yylineno++; }
.                   { printf("Unrecognized token%s!\n", yytext);
exit(1);  }

%%
```

2.2 Bison Code

Bison Program:

- Exploited WHILE code to do similar actions as to IF statement, without repeating the steps and not using the br label
- Extra comparisons above were added similarly to existing comparisons

parser.y

```
%{
	#include <stdio.h>
	#include <string.h>
	#include <stdlib.h>
	#include "data.h"
	extern FILE *yyin;
	extern int yylineno;
	extern char* yytext;
	int yylex();
	int yyerror(const char *p) { printf("Error : %s\n",p);
		printf("About line %d\n", yylineno);
		printf("Near %s\n", yytext);
		return 0;
 }

	void undefined(char *varname) { printf("Reference to 
undefined variable %s\n", varname); exit(1); }

	FILE *output;

	int comparecount = 0;
	int lblcount = 0;
	int nestingstack[10];
	int nestinglevel = 0;

	char* programname;
	char header[100];
	%}


%union {
	int ival;
	char* str_val;
};

%token <ival> DECLARE PROGRAM END SEMICOLON COMMA
%token <ival> LPAR RPAR
%token <ival> ASSIGN PRINT READ MSG NEWL READOCT PRINTOCT
%token <ival> PLUS MINUS MULTIPLY DIVIDE
%token <ival> LESS GE LE GT EQ NEQ
%token <ival> WHILE DO ENDWHILE
```

```
%token <ival> IF THEN ENDIF

%token <str_val> IDENTIFIER QSTRING
%token <ival> NUMBER

%type <ival> expression
%type <ival> term
%type <ival> factor

//Grammar for programming language
%%
program: PROGRAM pname declarations statements END {
fprintf(output,"exit\n"); printsyms(output); printstrings(output);
};

pname: IDENTIFIER SEMICOLON                                {
programname = $1;
     strcpy(header,"\"Program "); strcat(header,$1);
strcat(header,"\"");
     int ival = addstring(header);
     fprintf(output,".origin 2400\napplication: writeline\n
msg%d\ncall newline\n", ival);
}

declarations:
| DECLARE declarationlist SEMICOLON        {  /* printf("Parsed
declarations\n"); */ };

declarationlist:
declarationlist COMMA IDENTIFIER                {   insert($3);
}
| IDENTIFIER                                      {
insert($1); };

statements: statements  statement
| statement;

statement:
| NEWL SEMICOLON                                    {
fprintf(output,"call newline\n"); }
| MSG QSTRING SEMICOLON                           {   int ival =
addstring($2); fprintf(output,"writeline\nmsg%d\n", ival); }
| READ IDENTIFIER SEMICOLON                       {
fprintf(output,"call readint\nmov r0,%s\n",$2);   }
| PRINT expression SEMICOLON                        {
fprintf(output,"mov (sp)+,r0\ncall printint\n"); }
| READOCT IDENTIFIER SEMICOLON                        {
fprintf(output,"call readoct\nmov r0,%s\n",$2);   }
| PRINTOCT expression SEMICOLON                         {
fprintf(output,"mov (sp)+,r0\ncall printoct\n"); }
| IDENTIFIER ASSIGN expression SEMICOLON            {
     fprintf(output,"mov (sp)+,%s\n", $1);
```

```
      free($1);
}
| WHILE                                                     {
fprintf(output,"wh%d:  ", lblcount); nestingstack[nestinglevel++]
= lblcount; }
comparison
DO                                                    {
fprintf(output,"tst r0\nbeq ewh%d\n",lblcount);   lblcount++;}
statements ENDWHILE SEMICOLON                         { int lbl;
nestinglevel--; lbl = nestingstack[nestinglevel];
fprintf(output,"br wh%d\newh%d: ",lbl,lbl);};

| IF                                                   {
fprintf(output,"if%d:  ", lblcount); nestingstack[nestinglevel++]
= lblcount; }
comparison
THEN                                                    {
fprintf(output,"tst r0\nbeq eif%d\n",lblcount);   lblcount++;}
statements ENDIF SEMICOLON                         { int lbl;
nestinglevel--; lbl = nestingstack[nestinglevel];
fprintf(output,"eif%d: ",lbl);};

expression: term                                            {   }
| expression PLUS term                                  {
fprintf(output,"add (sp)+,(sp)\n"); }
| expression MINUS term                                 {
fprintf(output,"neg (sp)\nadd (sp)+,(sp)\n"); };

term: factor                                                {  }
| term MULTIPLY factor                                  {
fprintf(output,"mov (sp)+,r3\nmul (sp)+,r3\nmov r3,-(sp)\n"); }
| term DIVIDE factor                                    {
fprintf(output,"mov (sp)+,r0\n clr r2\nmov (sp)+,r3\n div
r0,r2\nmov r2,-(sp)\n");};

factor: LPAR expression RPAR                                {  }
| IDENTIFIER                                            {
fprintf(output,"mov %s,-(sp)\n", $1); }
| NUMBER                                                {
fprintf(output,"mov #%o,-(sp)\n", $1); };

comparison: LPAR expression LESS expression RPAR            {
      fprintf(output,"clr r0\n");
      fprintf(output,"cmp (sp)+,(sp)+\n");
      fprintf(output,"ble cmp%d\n",comparecount);
      fprintf(output,"inc r0\n");
      fprintf(output,"cmp%d: ",comparecount);
      comparecount++;
}
| LPAR expression GE expression RPAR                   {
      fprintf(output,"clr r0\n");
      fprintf(output,"cmp (sp)+,(sp)+\n");
```

```
        fprintf(output,"bgt cmp%d\n",comparecount);
        fprintf(output,"inc r0\n");
        fprintf(output,"cmp%d: ",comparecount);
        comparecount++;
}
| LPAR expression LE expression RPAR             {
        fprintf(output,"clr r0\n");
        fprintf(output,"cmp (sp)+,(sp)+\n");
        fprintf(output,"blt cmp%d\n",comparecount);
        fprintf(output,"inc r0\n");
        fprintf(output,"cmp%d: ",comparecount);
        comparecount++;
}
| LPAR expression GT expression RPAR             {
        fprintf(output,"clr r0\n");
        fprintf(output,"cmp (sp)+,(sp)+\n");
        fprintf(output,"bge cmp%d\n",comparecount);
        fprintf(output,"inc r0\n");
        fprintf(output,"cmp%d: ",comparecount);
        comparecount++;
}
| LPAR expression EQ expression RPAR             {
        fprintf(output,"clr r0\n");
        fprintf(output,"cmp (sp)+,(sp)+\n");
        fprintf(output,"bne cmp%d\n",comparecount);
        fprintf(output,"inc r0\n");
        fprintf(output,"cmp%d: ",comparecount);
        comparecount++;
}
| LPAR expression NEQ expression RPAR             {
        fprintf(output,"clr r0\n");
        fprintf(output,"cmp (sp)+,(sp)+\n");
        fprintf(output,"beq cmp%d\n",comparecount);
        fprintf(output,"inc r0\n");
        fprintf(output,"cmp%d: ",comparecount);
        comparecount++;
}

%%

void copyFilePDP(char* filename) {
        FILE* tmp = fopen(filename,"r");
        char ch;
        while(( ch = fgetc(tmp) ) != EOF ) fputc(ch,output);
        fclose(tmp);
}

int main(int argc, char** argv)
{
        char filename[128];
        printf("Enter name of file with program to be interpreted :
");
```

```
      scanf("%s",filename);
      FILE* input = fopen(filename,"r");
      if ( input == NULL ) { printf( "Could not open %s \n " ,
filename); exit (0);}
      yyin = input;
      output = fopen("pdp11.txt","w");
      copyFilePDP("os.txt");
      copyFilePDP("libc.txt");
      yyparse();
      fprintf(output,".end osstart\n");
      fclose(output);
      return 0;
}
```

2.4 Results

Example 1 - Given IF Tests

BASIC Code (Given in Assignment Spec):

```
PROGRAM iftest;
      VAR x;
      MSG "Conditional tests - comparing number with 3";
      NEWL;
      MSG "Enter value x = ";
      READ x;
      MSG "X < 3";
      IF(x<3)THEN
            MSG " - True";
      ENDIF;
      NEWL;
      MSG "x <= 3";
      IF(x<=3)THEN
            MSG " - True";
      ENDIF;
      NEWL;
      MSG "x == 3";
      IF(x==3)THEN
            MSG " - True";
      ENDIF;
      NEWL;
      MSG "x != 3";
      IF(x!=3)THEN
            MSG " - True";
      ENDIF;
      NEWL;
      MSG "x >= 3";
      IF(x>=3)THEN
            MSG " - True";
      ENDIF;
      NEWL;
      MSG "x > 3";
      IF(x>3)THEN
            MSG " - True";
      ENDIF;
      NEWL;
END
```

PDP-11 Assembly (Excluding OS Code):

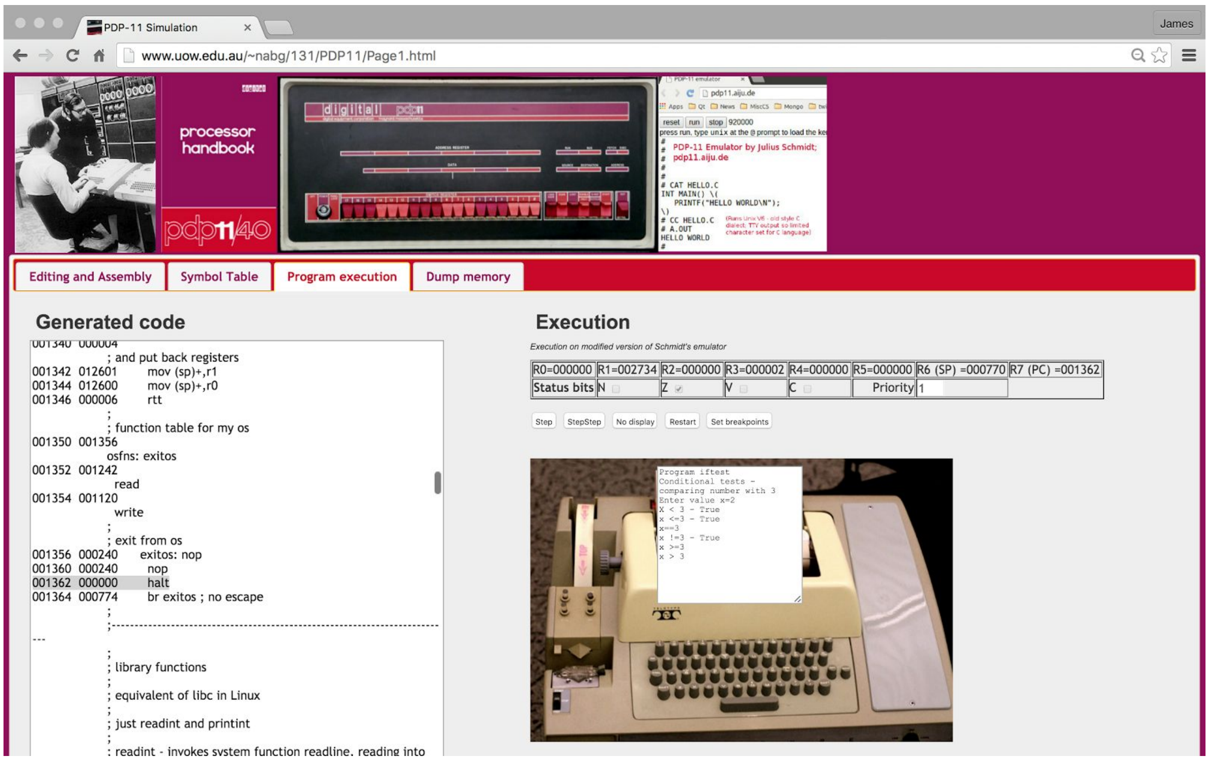
```
;----------------------------
.origin 2400
application: writeline
 msg0
call newline
writeline
msg1
call newline
writeline
msg2
call readint
mov r0,x
writeline
msg3
if0:  mov x,-(sp)
mov #3,-(sp)
clr r0
cmp (sp)+,(sp)+
ble cmp0
inc r0
cmp0: tst r0
beq eif0
writeline
msg4
eif0: call newline
writeline
msg5
if1:  mov x,-(sp)
mov #3,-(sp)
clr r0
cmp (sp)+,(sp)+
blt cmp1
inc r0
cmp1: tst r0
beq eif1
writeline
msg6
eif1: call newline
writeline
msg7
if2:  mov x,-(sp)
mov #3,-(sp)
clr r0
cmp (sp)+,(sp)+
bne cmp2
inc r0
cmp2: tst r0
beq eif2
writeline
msg8
```

```
eif2: call newline
writeline
msg9
if3:  mov x,-(sp)
mov #3,-(sp)
clr r0
cmp (sp)+,(sp)+
beq cmp3
inc r0
cmp3: tst r0
beq eif3
writeline
msg10
eif3: call newline
writeline
msg11
if4:  mov x,-(sp)
mov #3,-(sp)
clr r0
cmp (sp)+,(sp)+
bgt cmp4
inc r0
cmp4: tst r0
beq eif4
writeline
msg12
eif4: call newline
writeline
msg13
if5:  mov x,-(sp)
mov #3,-(sp)
clr r0
cmp (sp)+,(sp)+
bge cmp5
inc r0
cmp5: tst r0
beq eif5
writeline
msg14
eif5: call newline
exit
x: .word 0
msg0: .string "Program iftest"
msg1: .string "Conditional tests - comparing number with 3"
msg2: .string "Enter value x = "
msg3: .string "X < 3"
msg4: .string " - True"
msg5: .string "x <= 3"
msg6: .string " - True"
msg7: .string "x == 3"
msg8: .string " - True"
msg9: .string "x != 3"
```

```
msg10: .string " - True"
msg11: .string "x >= 3"
msg12: .string " - True"
msg13: .string "x > 3"
msg14: .string " - True"
.end osstart
```

PDP-11 Results:

Running:



Generated code

```
001340 000004
             ; and put back registers
001342 012601       mov (sp)+,r1
001344 012600       mov (sp)+,r0
001346 000006       rtt
             ;
             ; function table for my os
001350 001356
             osfns: exitos
001352 001242
              read
001354 001120
              write
             ;
             ; exit from os
001356 000240     exitos: nop
001360 000240       nop
001362 000000       halt
001364 000774       br exitos ; no escape
             ;
             ;------------------------------------------------------------
---
             ;
             ; library functions
             ;
             ; equivalent of libc in Linux
             ;
             ; just readint and printint
             ;
             ; readint - invokes system function readline, reading into
```

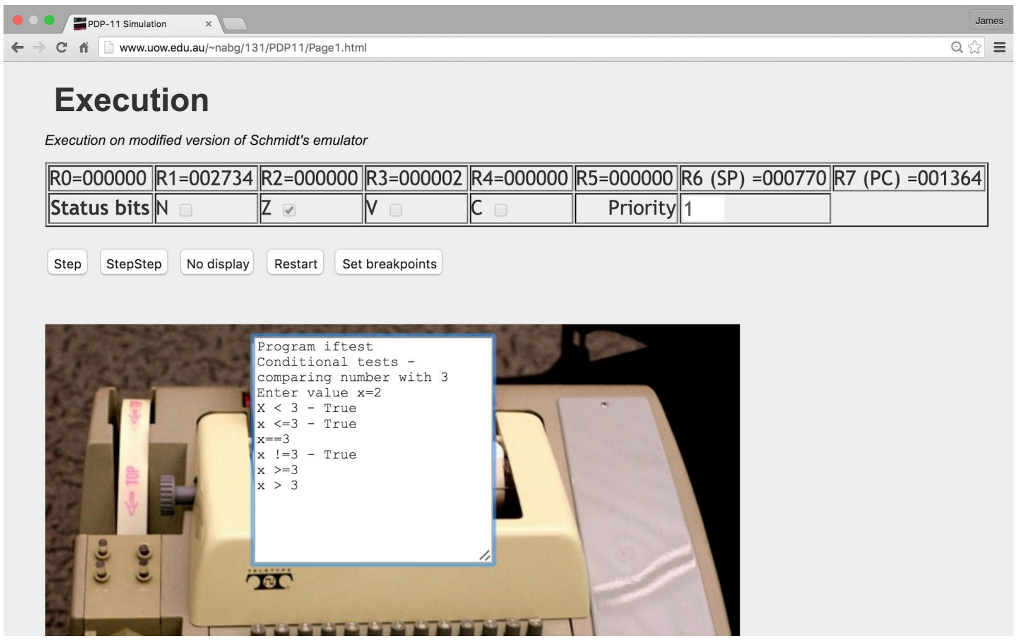
Execution

Execution on modified version of Schmidt's emulator

R0=000000	R1=002734	R2=000000	R3=000002	R4=000000	R5=000000	R6 (SP) =000770	R7 (PC) =001362
Status bits	N ☐	Z ✓	V ☐	C ☐	Priority	1	

```
Program iftest
Conditional tests -
comparing number with 3
Enter value x=2
X < 3 - True
x <=3 - True
x==3
x !=3 - True
x >=3
x > 3
```

Results where x = 2



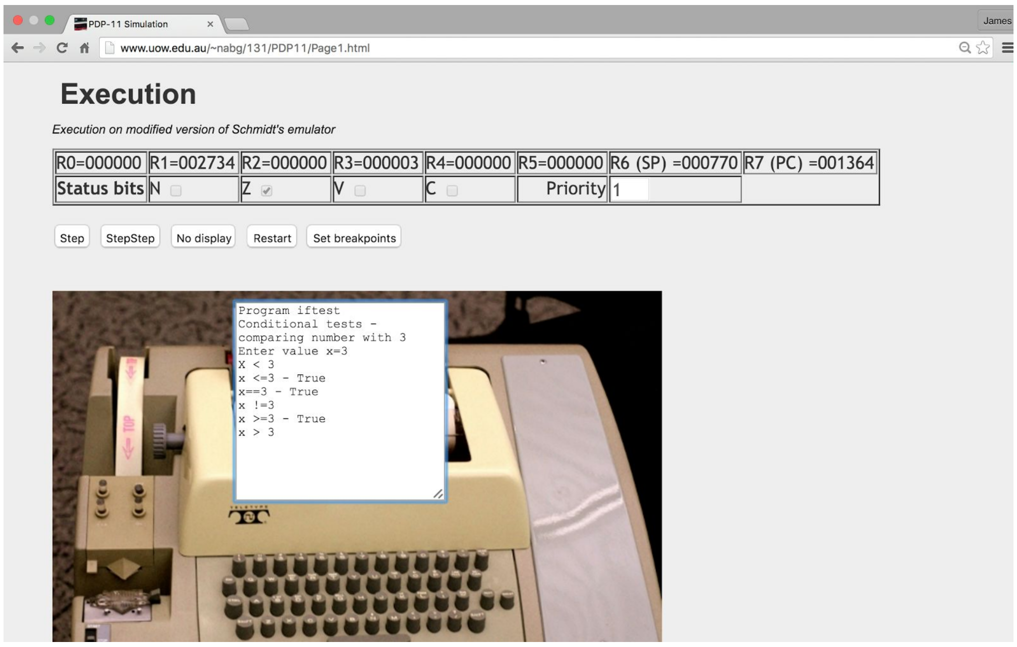
Execution

Execution on modified version of Schmidt's emulator

R0=000000	R1=002734	R2=000000	R3=000002	R4=000000	R5=000000	R6 (SP) =000770	R7 (PC) =001364
Status bits	N ☐	Z ✓	V ☐	C ☐	Priority	1	

```
Program iftest
Conditional tests -
comparing number with 3
Enter value x=2
X < 3 - True
x <=3 - True
x==3
x !=3 - True
x >=3
x > 3
```

Results where x = 3



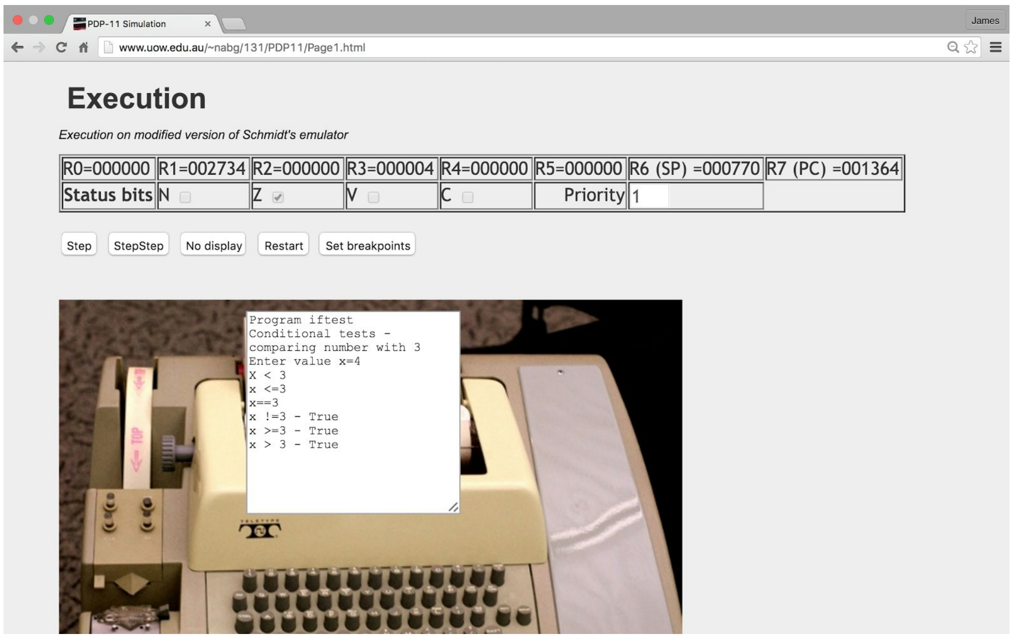
Execution

Execution on modified version of Schmidt's emulator

R0=000000	R1=002734	R2=000000	R3=000003	R4=000000	R5=000000	R6 (SP) =000770	R7 (PC) =001364
Status bits	N ☐	Z ✓	V ☐	C ☐	Priority	1	

```
Program iftest
Conditional tests -
comparing number with 3
Enter value x=3
X < 3
x <=3 - True
x==3 - True
x !=3
x >=3 - True
x > 3
```

Results where x = 4



Execution

Execution on modified version of Schmidt's emulator

R0=000000	R1=002734	R2=000000	R3=000004	R4=000000	R5=000000	R6 (SP) =000770	R7 (PC) =001364
Status bits	N ☐	Z ✓	V ☐	C ☐	Priority	1	

```
Program iftest
Conditional tests -
comparing number with 3
Enter value x=4
X < 3
x <=3
x==3
x !=3 - True
x >=3 - True
x > 3 - True
```

Example 2 - IF condition inside WHILE loop

BASIC Code

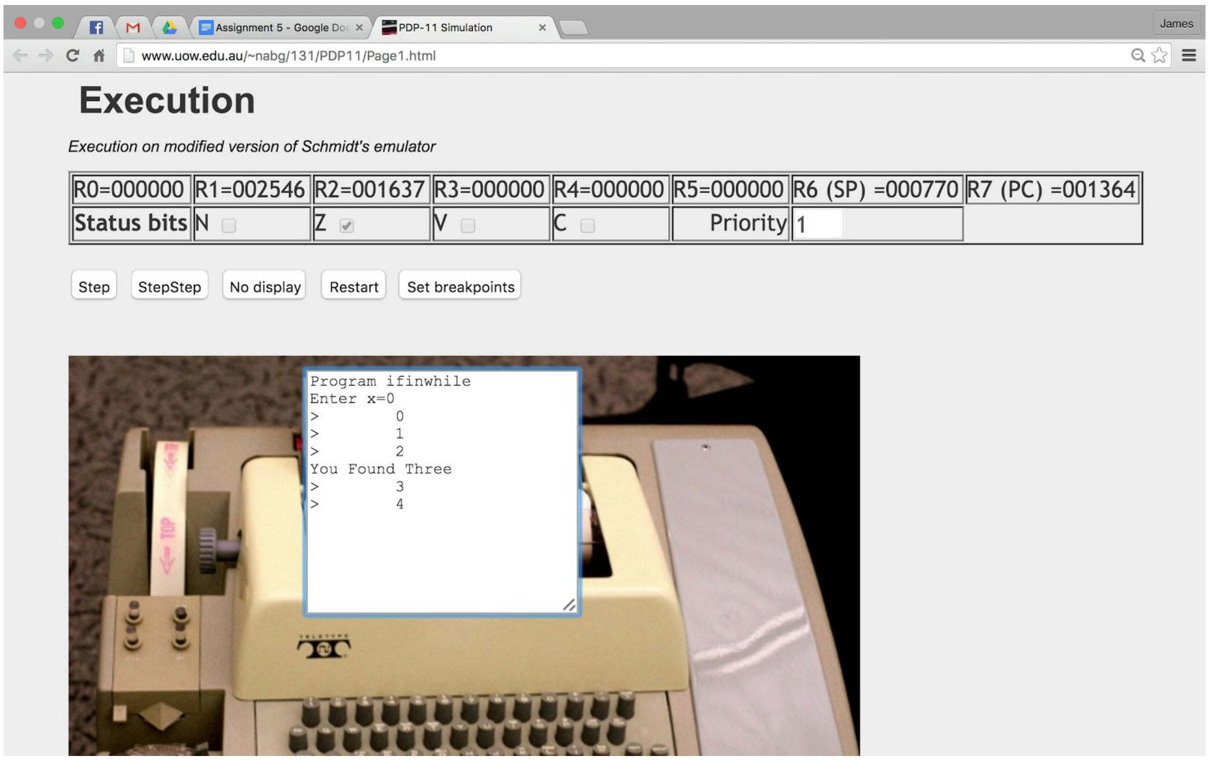
```
PROGRAM ifinwhile;
     VAR x;
     MSG "Enter x = ";
     READ x;
     WHILE(x<=4) DO
          MSG "> ";
          PRINT x;
          NEWL;
          IF(x==2)THEN
               MSG "You Found Three";
               NEWL;
          ENDIF;
          x=x+1;
     ENDWHILE;
END
```

PDP-11 Assembly (Excluding OS Code):

```
;----------------------------
.origin 2400
application: writeline
 msg0
call newline
writeline
msg1
call readint
mov r0,x
wh0:  mov x,-(sp)
mov #4,-(sp)
clr r0
cmp (sp)+,(sp)+
blt cmp0
inc r0
cmp0: tst r0
beq ewh0
writeline
msg2
mov x,-(sp)
mov (sp)+,r0
call printint
call newline
if1:  mov x,-(sp)
mov #2,-(sp)
clr r0
cmp (sp)+,(sp)+
bne cmp1
inc r0
cmp1: tst r0
beq eif1
writeline
msg3
call newline
eif1: mov x,-(sp)
mov #1,-(sp)
add (sp)+,(sp)
mov (sp)+,x
br wh0
ewh0: exit
x: .word 0
msg0: .string "Program ifinwhile"
msg1: .string "Enter x = "
msg2: .string "> "
msg3: .string "You Found Three"
.end osstart
```

PDP-11 Results

Using input x=0


Assignment 5 - Google Doc
PDP-11 Simulation
James
www.uow.edu.au/~nabg/131/PDP11/Page1.html
Step
StepStep
No display
Restart
Set breakpoints

Execution

Execution on modified version of Schmidt's emulator

R0=000000	R1=002546	R2=001637	R3=000000	R4=000000	R5=000000	R6 (SP) =000770	R7 (PC) =001364
Status bits	N ☐	Z ✓	V ☐	C ☐	Priority	1	

```
Program ifinwhile
Enter x=0
>         0
>         1
>         2
You Found Three
>         3
>         4
```

Example 3 - WHILE loop inside IF condition

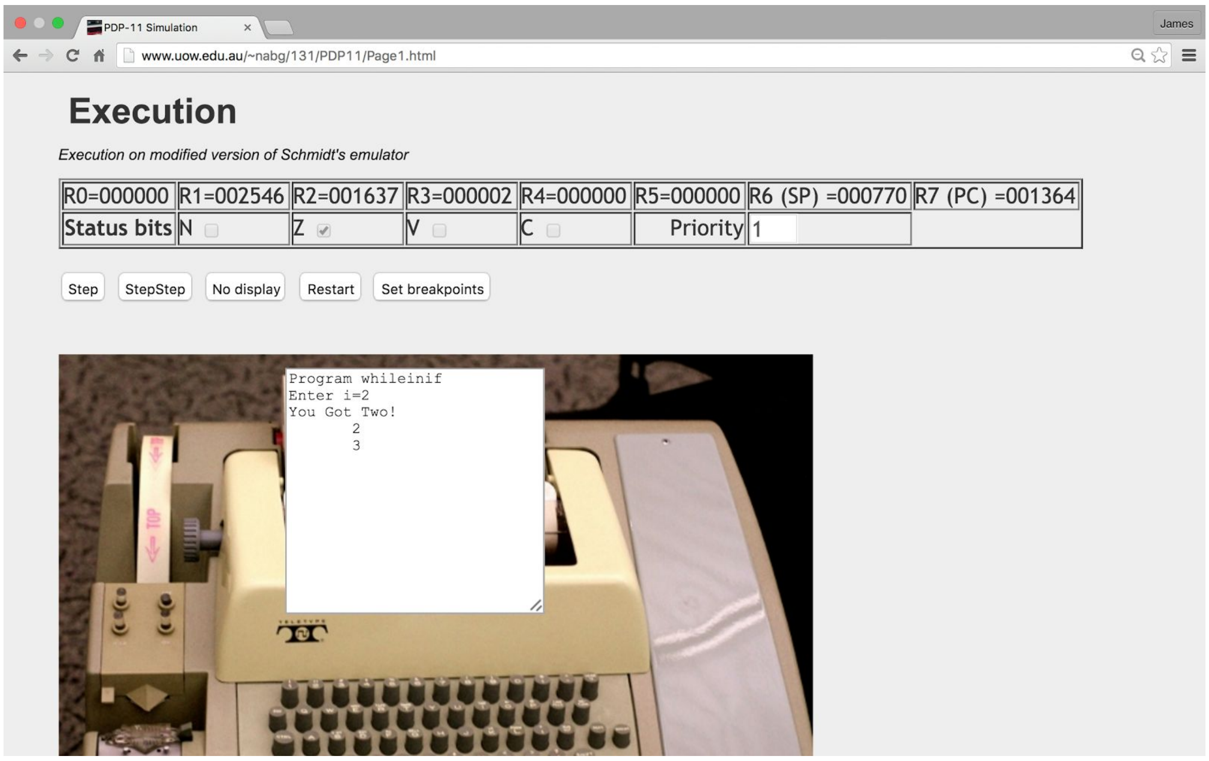
BASIC Code

```
PROGRAM whileinif;
      VAR i;
      MSG "Enter i = ";
      READ i;
      IF(i==2)THEN
            MSG "You Got Two!";
            NEWL;
            WHILE (i<4) DO
                  PRINT i;
                  NEWL;
                  i=i+1;
            ENDWHILE;
      ENDIF;
      NEWL;
END
```

PDP-11 Assembly (Excluding OS Code):

```
;----------------------------
.origin 2400
application: writeline
 msg0
call newline
writeline
msg1
call readint
mov r0,i
if0:  mov i,-(sp)
mov #2,-(sp)
clr r0
cmp (sp)+,(sp)+
bne cmp0
inc r0
cmp0: tst r0
beq eif0
writeline
msg2
call newline
wh1:  mov i,-(sp)
mov #4,-(sp)
clr r0
cmp (sp)+,(sp)+
ble cmp1
inc r0
cmp1: tst r0
beq ewh1
mov i,-(sp)
mov (sp)+,r0
call printint
call newline
mov i,-(sp)
mov #1,-(sp)
add (sp)+,(sp)
mov (sp)+,i
br wh1
ewh1: eif0: call newline
exit
i: .word 0
msg0: .string "Program whileinif"
msg1: .string "Enter i = "
msg2: .string "You Got Two!"
.end osstart
```

Results where i = 2



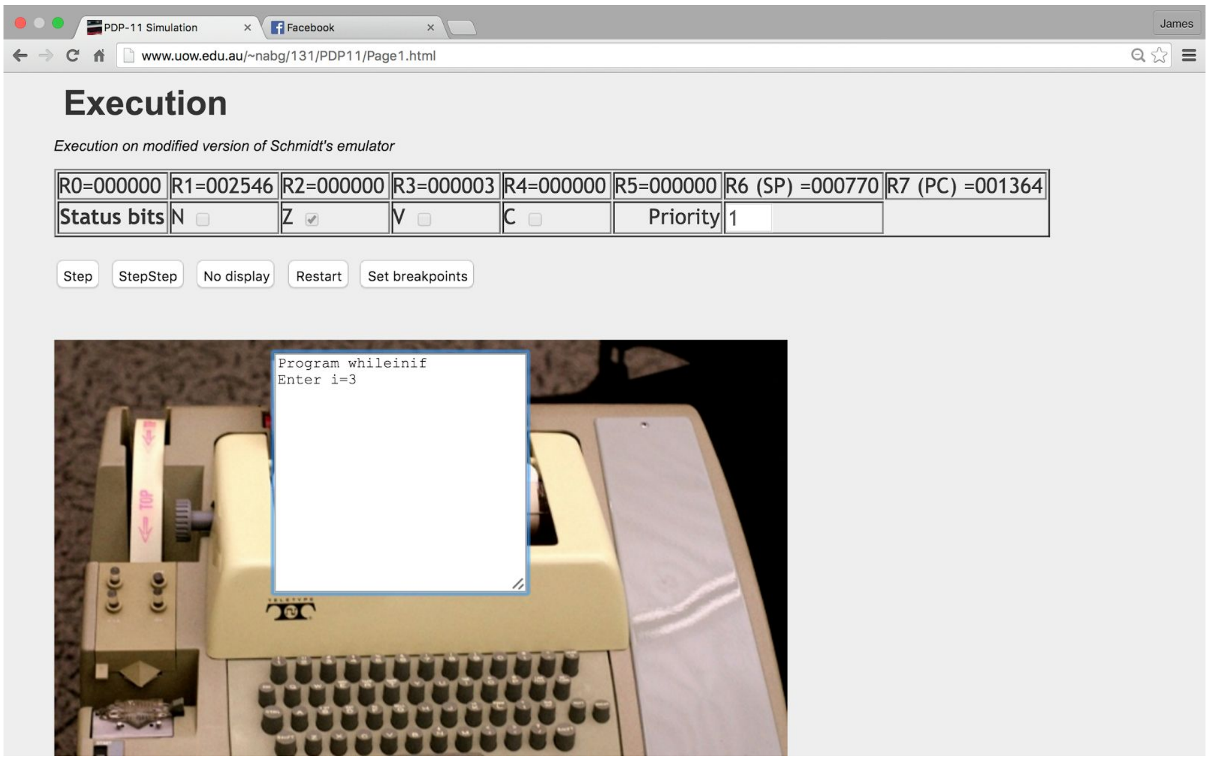
Execution

Execution on modified version of Schmidt's emulator

R0=000000	R1=002546	R2=001637	R3=000002	R4=000000	R5=000000	R6 (SP) =000770	R7 (PC) =001364
Status bits	N ☐	Z ✓	V ☐	C ☐	Priority	1	

```
Program whileinif
Enter i=2
You Got Two!
        2
        3
```

Results where i = 3



Execution

Execution on modified version of Schmidt's emulator

R0=000000	R1=002546	R2=000000	R3=000003	R4=000000	R5=000000	R6 (SP) =000770	R7 (PC) =001364
Status bits	N ☐	Z ✓	V ☐	C ☐	Priority	1	

```
Program whileinif
Enter i=3
```

3. Conclusion

As we can see the BASIC code interprets to PDP-11 Assembly and compiles as expected on the PDP-11.