Design Requirements

- Use Logisim to Implement a set of instructions and the given circuit including a pipeline for Fetching, Decoding, and Executing our Instructions; Include the design of a Finite State Machine in our control unit logic to drive the pipeline.
- Design a piece of software to embed using our instruction set.
- Use an 8051 microcontroller simulator to perform the exact same task as our machine's embedded code; or some alternative assembly language execution (ARM, Motorola, etc.) Microcontroller or Microprocessor.
- Compare the performance of our machine to that of at least one existing assembly language.

Op Codes

```
      00h (OP-CODE = 00000000) Ri + counter#1 → Rk
      01h (OP-CODE = 00000001) Ri + counter#2 → Rk
      02h (OP-CODE = 00000010) Ri + Rj → Rk
      03h (OP-CODE = 00000011) counter#1 + counter#2 → Rk

04h to 07h (OP-CODE = 000001XX) Reserved for subtraction instructions

      08h (OP-CODE = 00001000) Ri x counter#1 → Rk, overflow → Rk+1
      09h (OP-CODE = 00001001) Ri x counter#2 → Rk, overflow → Rk+1
      0Ah (OP-CODE = 00001010) Ri x Rj → Rk, overflow → Rk+1
      0Bh (OP-CODE = 00001011) counter#1 x counter#2 →Rk, overflow→Rk+1

0Ch to 0Fh (OP-CODE = 000011XX) Reserved for division instructions

      10h (OP-CODE = 00010000) Compare Ri with counter#1 → Rk
      11h (OP-CODE = 00010001) Compare Ri with counter#2 → Rk
      12h (OP-CODE = 00010010) Compare Ri with Rj → Rk
      13h (OP-CODE = 00010011) Compare Counters

      14h (OP-CODE = 00010100) Ri AND counter#1 → Rk
      15h (OP-CODE = 00010101) Ri AND counter#2 → Rk
      16h (OP-CODE = 00010110) Ri AND Rj → Rk
      17h (OP-CODE = 00010111) AND counters → Rk

      18h (OP-CODE = 00011000) Ri OR counter#1 → Rk
      19h (OP-CODE = 00011001) Ri OR counter#2 → Rk
      1Ah (OP-CODE = 00011010) Ri OR Rj → Rk
      1Bh (OP-CODE = 00011011) OR counters→ Rk

1Ch to 1Fh (OP-CODE = 00011XXX) Reserved for future instructions
20h to FFh                    Reserved for future instructions
```

Simplified Functions

Ma1 = AC

Mb1 = ABC + E

Ma0 = ABC + DE

Mb0 = ABC + D'E'

Da2 = A'B + BC + AB'C'

Da1 = A'B' + B'C' + ABC

Da0 = AB' + B'C + A'BC'

Db2 = AB'C' + BC + A'B

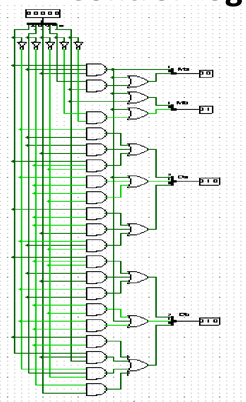
Db1 = B'C' + ABC + A'B'

Db0 = AB' + ACE + A'BC' + B'C

Truth Table for Control Logic

		A	B	C	D	E	Ma1	Ma0	Mb1	Mb0	Da2	Da1	Da0	Db2	Db1	Db0
ADD	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0
	2	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0
	3	0	0	0	1	1	0	1	1	0	0	1	0	0	1	0
SUB	4	0	0	1	0	0	0	0	0	1	0	1	1	0	1	1
	5	0	0	1	0	1	0	0	1	0	0	1	1	0	1	1
	6	0	0	1	1	0	0	0	0	0	0	1	1	0	1	1
	7	0	0	1	1	1	0	1	1	0	0	1	1	0	1	1
MULT	8	0	1	0	0	0	0	0	0	1	1	0	1	1	0	1
	9	0	1	0	0	1	0	0	1	0	1	0	1	1	0	1
	10	0	1	0	1	0	0	0	0	0	1	0	1	1	0	1
	11	0	1	0	1	1	0	1	1	0	1	0	1	1	0	1
DIV	12	0	1	1	0	0	0	0	0	1	1	0	0	1	0	0
	13	0	1	1	0	1	0	0	1	0	1	0	0	1	0	0
	14	0	1	1	1	0	0	0	0	0	1	0	0	1	0	0
	15	0	1	1	1	1	0	1	1	0	1	0	0	1	0	0
COMP	16	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1
	17	1	0	0	0	1	0	0	1	0	1	1	1	1	1	1
	18	1	0	0	1	0	0	0	0	0	1	1	1	1	1	1
	19	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1
AND	20	1	0	1	0	0	0	0	0	1	0	0	1	0	0	1
	21	1	0	1	0	1	0	0	1	0	0	0	1	0	0	1
	22	1	0	1	1	0	0	0	0	0	0	0	1	0	0	1
	23	1	0	1	1	1	0	1	1	0	0	0	1	0	0	1
OR	24	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0
	25	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0
	26	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	27	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0
STORE	28	1	1	1	0	0	1	1	1	1	1	1	0	1	1	0
ROM	29	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1
X	30	1	1	1	1	0	X	X	X	X	X	X	X	X	X	X
X	31	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X

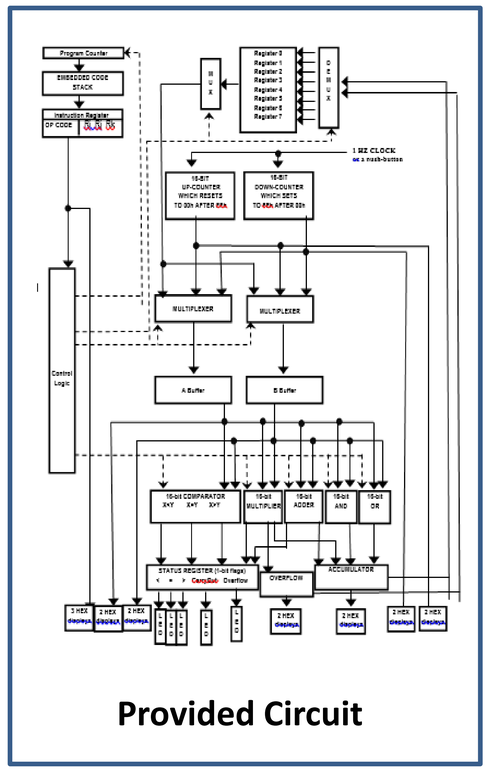
Control Logic



Stored in ROM

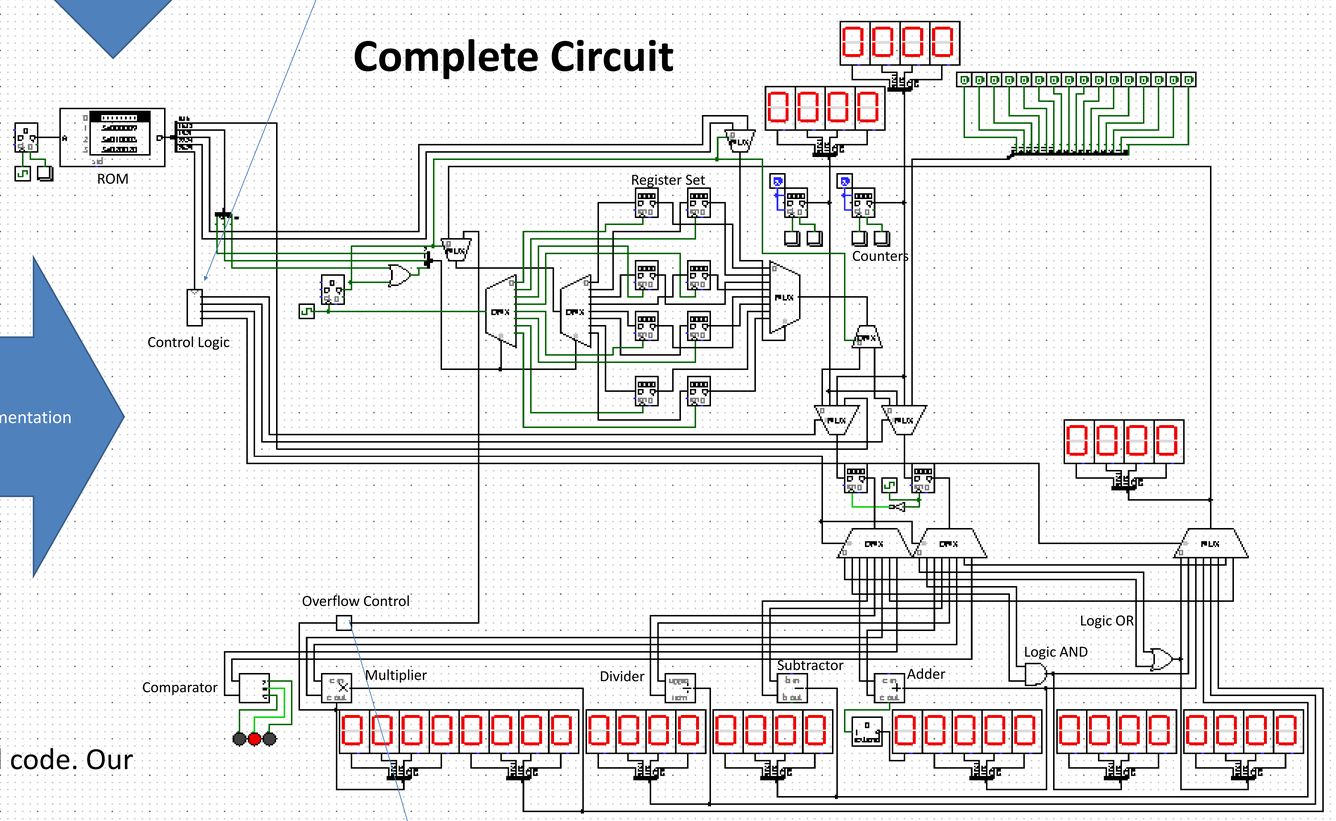
Intel Assembly Code for Celsius to Fahrenheit

```
    .file   "convert.c"
    .section    .rodata
.LC0:
    .string "%d"
.LC1:
    .string "%d\n"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $.LC0, %eax
    leaq    -8(%rbp), %rdx
    movq    %rdx, %rsi
    movq    %rax, %rdi
    movl    $0, %eax
    call    __isoc99_scanf
    movl    -8(%rbp), %edx
    movl    %edx, %eax
    sall    $3, %eax
    leal    (%rax,%rdx), %ecx
    movl    $1717986919, %edx
    movl    %ecx, %eax
    imull   %edx
    sarl    %edx
    movl    %ecx, %eax
    sarl    $31, %eax
    movl    %edx, %ecx
    subl    %eax, %ecx
    movl    %ecx, %eax
    addl    $32, %eax
    movl    %eax, -4(%rbp)
    movl    $.LC1, %eax
    movl    -4(%rbp), %edx
    movl    %edx, %esi
    movq    %rax, %rdi
    movl    $0, %eax
    call    printf
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
    .section    .note.GNU-stack,"",@progbits
```



Provided Circuit

Implementation

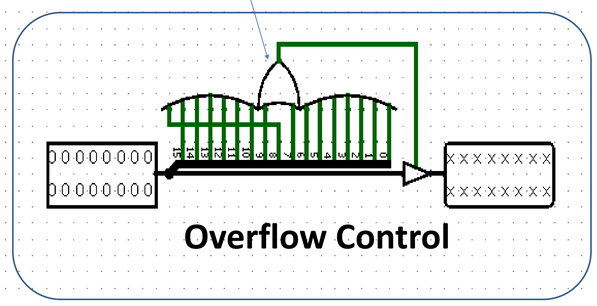


Complete Circuit

We used the Intel x86 Assembly language to perform the task as our machine's embedded code. Our program can successfully convert a Celsius value to Fahrenheit.
Below are the opcodes of our program converted into hexadecimal for Logisim.

ROM embedded Code for Celsius to Fahrenheit

00000000 3a000009 3a010005 3a020020
38030000 14c40000 1c650000 05540000
00000000



Overflow Control

Computer Design Competition

Tucker Strausbaugh
Sajid Amir
Kyler Killinger
EGR 433: Advanced Computer Engineering
Elizabethtown College, Elizabethtown, PA