

Keywords: Minimal Instruction Set, 4-bit opcodes, MISC, RISC CPU

Draft

Presenting a virtual machine after the von Neumann model, composed of seven registers, an instruction set of sixteen instructions, each represented by a four bit operation code, and a method for encoding these operation codes in a 32 bit instruction word so that eight instructions are loaded during each fetch cycle of the machine, based on [Moore].

0. Examples

```
-----
```

CODE	DESCRIPTION
Z	Push 0
Z NOT	Push -1
Z NOT PLUS	DEC
Z INC	Push 1
NOT INC	Two's complement (NEG)
NOT INC PLUS	Subtract
SNAP 0	Data Pointer
SNAP 0 LD 5	Forth 5 PICK
SNAP 1 LD -1	Pop
SNAP -1 ST 0	Push
ENTRY 1	Get a parameter
SWAP	Forth SWAP
Z COPY	Forth DUP
Z MSB	Minus Flag
Z MSB <offs> SKIPZ	Branch if negative
Z SWAP MSB	Logical shift right (Discard S)
Z COPY MSB	Arithmetic shift right (Discard S)
NOT AND EOR	Inclusive OR (IOR)
Z SKIPZ <xxx>	Conditional branch
	<xxx> only if R!=0, else next instruction
ENTRY 4 SKIPZ	Relative branch if zero, using offset in Parameter 4

1. Internal Registers

1.1 System Registers

1.1.1 Instruction Pointer Register (P)

The P register is a program counter used to store the address of the next instruction in memory. After decoding is complete, a new instruction word is fetched from the address in the P register. The P register is incremented by one before the first instruction slot is decoded. Modifying this register alters the flow of control.

1.1.2 Instruction Register (I)

During Fetch, the I register is loaded with the instruction word stored in the memory address in the P register. The instruction word consists of eight consecutive groups of four bits called instruction slots, each encoding one operation code or a four-bit direct operand.

1.1.3 Data Pointer Register (D)

The D register holds a memory address which points to a memory cell in the center of a dynamic data structure containing process specific data, to which reference is made by addressing relative to D. The D pointer value can be adjusted incrementally within the address space, similar to the working of a common stack during push/pop operations, during which the stack pointer is incremented/decremented by one.

1.1.4 Entry Register (E)

The E register is set during an absolute branch instruction to the branch target address, typically the entry point of a subroutine or other self-contained code fragment. By establishing the practice of storing literal values including branch targets directly before such an entry point, these literals can be accessed by means of addressing relative to E. The relative branch instruction SKIPZ does not alter E, and so within a subroutine, where mostly relative branches are used, E is a reliable means for provisioning literals to the running process.

1.2 Register Stack

1.2.1 Top-of-Stack Register (T)

The value of this register is the implied primary operand for all instructions. Any instruction that overwrites the content of T places a copy of the old value in the C register, overwriting its value. T and S can be swapped.

1.2.2 Second-on-Stack Register (S)

The value in this register is the implied secondary operand for instructions using two operands. When the operand value in S is consumed during a binary operation, a copy of C is placed in S. Instructions that push a new value into T overwrite T by the old value of S (during a push, T spills into S). T and S can be swapped.

1.2.3 Copy Register (C)

During instructions that overwrite the value of T, the previous value of T is copied into the C register, overwriting its value.

2. Decoding and Executing Instruction Words

We will now describe the method of decoding the eight instruction slots from a 32-bit instruction word and of executing each operation code in turn.

The Instruction Register (I) contains a 32-bit instruction word, organized into a sequence of eight four-bit groups called slots. The least significant four-bit group (slot) of the instruction word is removed by shifting I four places to the right, replacing the high order bits by the value 0.

The slot shifted out is executed and another shift cycle occurs, until either the instruction word becomes zero, or the conditional relative branch instruction SKIPZ is executed while T=0. In both cases, a fetch occurs and (I) is reloaded with the new instruction word.

3. Instruction Set

SKIPZ	ST-n	Z	MSB
SNAP-n	LD-n	INC	AND
ENTRY-n	SWAP	NOT	EOR
GO	COPY	SHL	PLUS

We will now describe the sixteen instructions
(S before, T before -- S after, T after)

3.1 Branch and Pointer Group

3.1.1 SKIPZ (FLAG OFFS --)

Conditional relative branch (Skip relative if zero). If FLAG is zero, discard the remaining slots of the current instruction word, add OFFS to P and initiate a new fetch. No effect on C.

3.1.2 SNAP-n (-- D_NEW)

Incrementally modify the D pointer inside a dynamic datastructure and push the new value onto the register stack. The SNAP opcode is followed by a slot encoding a signed octal digit n (sign, b1 b1 b3). Add n to D and push D into T, spilling into S. No effect on C.

3.1.3 ENTRY-n (-- E)

The ENTRY opcode is followed by a slot encoding a hexadecimal digit n (sign, b1 b1 b3). Load memory M[E-n-1] and push the new value onto the register stack. No effect on C.

3.1.4 GO (ADDR -- P_old)

Unconditional absolute jump to ADDR. Set E to ADDR. Set P to ADDR. Push old value of P onto stack. No effect on C.

3.2 Memory and Stack Group

3.2.1 ST-n (VAL ADDR --)

Store T into a memory cell.

The ST opcode is followed by a slot encoding the signed octal offset n (sign, b1 b1 b3). Store VAL into memory cell M[ADDR+n]. No effect on C.

3.2.2 LD-n (ADDR -- VAL)

The LD opcode is followed by a slot encoding the signed octal offset n (sign, b1 b1 b3).

Push content of memory cell M[ADDR+n] into T. Copy old value of T to C.

3.2.3 SWAP (VAL1 VAL2 -- VAL2 VAL1)

Exchange T and S. No effect on C.

3.2.4 COPY (VAL1 X - VAL1 VAL1)

Copy S into T. No effect on C.

3.3 Unary Group

3.3.1 Z (-- 0)

Push a zero onto the stack.

Copy old value of T to C.

3.3.2 INC (VAL -- VAL+1)

Copy T to C, increment VAL by 1.

3.3.3 NOT (VAL -- NOT VAL)

Copy T to C, invert bits of VAL.

3.3.4 SHL (VAL -- VAL<<1)

Logical Shift Left

Copy T to C, shift T left by 1.

3.4 Binary Group

3.4.1 MSB (VAL1 VAL2 - VAL1 VAL2>>1 SIGN EXTEND WITH MSB OF VAL1)

Arithmetic Shift Right T using MSB of S

3.4.2 AND / EOR / PLUS (VAL1 VAL2 -- VAL3)

Compute a binary logic function. Copy C to temp. Copy T to C, store function result in T, copy temp to S.