

Nibz Instruction Set

Introduction

All instructions are detailed below. They begin with a heading detailing the instruction mnemonic, and the decimal value of the operation code. This is then followed by the register transfer language description of the actual functioning of the instruction. A short description then follows.

Instruction: (Advanced Branch) ≥ 16

RTL: $R:=R-1, [R]:=P, P:=IR$

This instruction performs a subroutine call. There is no operation code overhead as the instruction register (IR) contains the jump address. This does mean however, that it is impossible to jump to any address below 16 using an advanced branch instruction.

Instruction: BA(BAck) :=0

RTL: $P:=[R], R:=R+1$

This instruction performs a subroutine return.

Instruction: FI (Fetch In) :=1

RTL: $A:=[Q], Q:=Q+1$

This instruction is mainly used for fetching memory variables and input output from ports which are memory mapped. The post increment is kept for the potential utility for copy reversal of data.

Instruction: RI (Return In) :=2

RTL: $Q=[R], R:=R+1$

This instruction is to allow manipulation of the return (R) stack. The load register Q is used as code becomes more efficient if A is not used.

Instruction: SI (Stack In) :=3

RTL: $A:=[S], S:=S+1$

This instruction loads a value from stack into the main working accumulator.

Instruction: DI (Difference) :=4

RTL: $A:=A \text{ XOR } [S], S:=S+1$

This instruction performs an exclusive or with an item on stack with the contents of the working register A.

Instruction: FA (Fetch Address) :=5

RTL: $Q:=[S], S:=S+1$

This instruction fetches the address stack item, to later load from memory.

Instruction: RA (Return Address) :=6

RTL: $R:=[S], S:=S+1$

This instruction loads the return stack pointer.

Instruction: SA (Stack Address) := 7

RTL: $S:=[S]$

This instruction loads the data stack pointer.

Instruction: BO (Both) :=8

RTL: $A:=(A \text{ AND } [S])*2+c, S:=S+1$

This instruction is the most complex to understand. It performs an and of data from the stack with the working register. The result is then rolled left one bit position through the carry bit. This may seem odd, but it reduces the area and logical complexity of the arithmetic and logic unit (ALU) while keeping addition as one of the major operations.

Instruction: FO (Fetch Out) :=9

RTL: $Q:=Q-1, [Q]:=A$

This instruction is mainly used for writing to memory variables, and to input or output ports.

Instruction: RO (Return Out) :=10

RTL: $R:=R-1, [R]:=Q$

This instruction is mainly used for return stack manipulation, and is the reverse of RI.

Instruction: SO (Stack Out) :=11

RTL: $S:=S-1, [S]:=A$

This instruction places the working register A on the stack.

Instruction: SU (SUM) :=12

RTL: $A:=A+[S]+c, S:=S+1$

This instruction performs an addition with carry of the working register and a stack item.

Instruction: FE (Fetch End) :=13

RTL: $S:=S-1, [S]:=Q$

This instruction saves the fetch pointer address or value in Q to the stack

Instruction: RE (Return End) :=14

RTL: $S:=S-1, [S]:=R$

This instruction places the return stack pointer onto the stack. It is very infrequently used. Even in multi-tasking systems it is not essential.

Instruction: SE (Stack End) :=15

RTL: $S:=S-1, [S]:=S$

This instruction saves the return stack pointer onto the stack. It also is very infrequently used.