

Pep/10

J. Stanley Warford

March 19, 2019

Here are the differences between Pep/9 and Pep/8 along with a rationale for each change.

1. RETn replaced by RET

In Pep/8, there are eight versions of the return statement, RET0, RET1, ..., RET7. RETn deallocates n bytes from the run-time stack, and then executes a return from a function call. The reasoning is that returns are always preceded by a deallocation of local variables, so programs are shorter if the assembly language programmer is not required to write an ADDSP instruction to explicitly deallocate the locals before the return.

While this ISA design may be justified on architectural principles, it turned out to be deficient pedagogically. The problem is that students must learn two different concepts, the deallocation mechanism for data and the return mechanism for flow of control. Combining these two different concepts into one statement can be confusing during the learning process. Pep/9 now requires students to explicitly deallocate locals with the ADDSP statement. An added stylistic advantage is that the explicit ADDSP to deallocate locals at the end of a function corresponds directly to the SUBSP at the beginning of the function to allocate locals.

2. Memory-mapped I/O

Of all the instructions in the Pep/8 instruction set, the most unrealistic are CHARI and CHARO for character input and output. Most real computer systems map input and output ports to main memory, which is now the design of Pep/9. In the new instruction set, there are no native input and output instructions. Instead, the Pep/8 instruction

```
CHARI alpha,ad
```

is replaced by the Pep/9 instructions

```
LDBA charIn,d    ;Load byte to A from the input port charIn
STBA alpha,ad    ;Store byte from A to alpha
```

and the Pep/8 instruction

```
CHARO beta,ad
```

is replaced by the Pep/9 instructions

```
LDBA beta,ad     ;Load byte to A from beta
STBA charOut,d   ;Store byte to the output port charOut
```

In the above code fragments, ad represents any valid addressing mode for the instruction. Symbols charIn and charOut are defined in the Pep/9 operating system and stored as machine vectors at the bottom of memory. Their values are included automatically in the symbol table of the assembler.

One disadvantage of memory-mapped I/O is that every CHARI and CHARO statement in a Pep/8 program must now be written as two statements, making programs longer. This disadvantage is mitigated by the fact that the trap instructions DECI, DECO, and STRO work as before, as the native I/O statements are hidden inside their trap routines.

The advantage is that students learn first hand how memory-mapped I/O works by loading from the input port and storing to the output port. This requirement also illustrates the concept and the use of the memory map, a topic students have a tendency to avoid with Pep/8. There is also a nice connection with the example in Chapter 11 on address decoding that shows how to wire an 8-port I/O chip into the memory map.

3. New native instruction CPBr

In Pep/8, byte quantities must be compared with CP_r, which compares two-byte quantities. Consequently, the high-order byte of the comparison must be considered, sometimes by clearing the high-order byte of the register before the comparison is made. The resulting assembler code for doing byte comparisons is convoluted.

CPBr is a new compare byte instruction that sets the status bits without regard to the high-order byte of the register. The resulting code is simpler to understand and to write.

The Register Transfer Language (RTL) specification of the Pep/8 load byte instruction LDBYTE_r is

$$r(8..15) \leftarrow \text{byte Oprnd} ; N \leftarrow r < 0, Z \leftarrow r = 0$$

The RTL specification of the Pep/9 load byte instruction, now named LDBr, is

$$r\langle 8..15 \rangle \leftarrow \text{byte Oprnd} ; N \leftarrow 0, Z \leftarrow r\langle 8..15 \rangle = 0$$

The N and Z bits are now set according to the properties of the byte quantity, which is always considered to be nonnegative, *i.e.*, unsigned. This specification is consistent with the fact that byte comparisons are always made with ASCII character values, not numeric values, and so produce results that would naturally be expected. It also simplifies the microcode implementation of the instruction in Chapter 12.

4. Improved mnemonics

Pep/9 renames the mnemonics for the compare, load, and store instructions as shown below.

Instruction	Pep/9	Pep/8
Compare word	CPWr	CPr
Compare byte	CPBr	<i>Not available</i>
Load word	LDWr	LDr
Load byte	LDBr	LDBYTEr
Store word	STWr	STr
Store byte	STBr	STBYTEr

Pep/9 retains the letters CP for compare, LD for load, and ST for store, but is now consistent in using the letters W for word, which is now required, and B for byte with this group of instructions. Not only is this naming convention more consistent, but there is a tendency for students to forget the meaning of a word (two bytes in the Pep computers). Including the letter W in the mnemonics for the two-byte instructions reinforces the meaning of “word”.

5. New trap instruction HEXO

Pep/9 eliminates the NOP2 and NOP3 trap instructions from the instruction set, which, together with the elimination of the RETn and character I/O instructions, allows the inclusion of another nonunary trap instruction. HEXO, which stands for hexadecimal output, was available in Pep/7 and is resurrected in Pep/9. It outputs a word as four hexadecimal characters.

6. Addressing mode nomenclature

Pep/9 changes the name “stack-indexed deferred” addressing to “stack-deferred indexed” addressing and the corresponding assembler notation from *sxf* to *sfx*. This change more accurately reflects the semantics of the addressing mode as the stack deferred operation happens *before* the index operation.

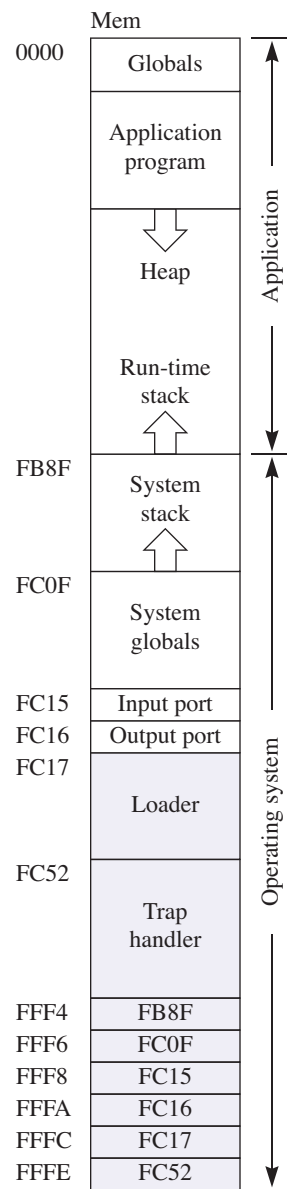
Here is the Pep/9 instruction set.

Instruction Specifier	Mnemonic	Instruction	Addressing Mode	Status Bits
0000 0000	STOP	Stop execution	U	
0000 0001	RET	Return from CALL	U	
0000 0010	RETR	Return from trap	U	
0000 0011	MOVSPA	Move SP to A	U	
0000 0100	MOVFLGA	Move NZVC flags to A<12..15>	U	
0000 0101	MOVAFLG	Move A<12..15> to NZVC flags	U	
0000 011r	NOTr	Bitwise invert r	U	NZ
0000 100r	NEGr	Negate r	U	NZV
0000 101r	ASLr	Arithmetic shift left r	U	NZVC
0000 110r	ASRr	Arithmetic shift right r	U	NZC
0000 111r	ROLr	Rotate left r	U	C
0001 000r	RORr	Rotate right r	U	C
0001 001a	BR	Branch unconditional	i, x	
0001 010a	BRLE	Branch if less than or equal to	i, x	
0001 011a	BRLT	Branch if less than	i, x	
0001 100a	BREQ	Branch if equal to	i, x	
0001 101a	BRNE	Branch if not equal to	i, x	
0001 110a	BRGE	Branch if greater than or equal to	i, x	
0001 111a	BRGT	Branch if greater than	i, x	
0010 000a	BRV	Branch if V	i, x	
0010 001a	BRC	Branch if C	i, x	
0010 010a	CALL	Call subroutine	i, x	
0010 011n	NOPn	Unary no operation trap	U	
0010 1aaa	NOP	Nonunary no operation trap	i	
0011 0aaa	DECI	Decimal input trap	d, n, s, sf, x, sx, sfx	NZV
0011 1aaa	DECO	Decimal output trap	i, d, n, s, sf, x, sx, sfx	
0100 0aaa	HEXO	Hexadecimal output trap	i, d, n, s, sf, x, sx, sfx	
0100 1aaa	STRO	String output trap	d, n, sf, x	
0101 0aaa	ADDSP	Add to stack pointer (SP)	i, d, n, s, sf, x, sx, sfx	NZVC
0101 1aaa	SUBSP	Subtract from stack pointer (SP)	i, d, n, s, sf, x, sx, sfx	NZVC
0110 raaa	ADDR	Add to r	i, d, n, s, sf, x, sx, sfx	NZVC
0111 raaa	SUBR	Subtract from r	i, d, n, s, sf, x, sx, sfx	NZVC
1000 raaa	ANDR	Bitwise AND to r	i, d, n, s, sf, x, sx, sfx	NZ
1001 raaa	ORR	Bitwise OR to r	i, d, n, s, sf, x, sx, sfx	NZ
1010 raaa	CPWR	Compare word to r	i, d, n, s, sf, x, sx, sfx	NZVC
1011 raaa	CPBR	Compare byte to r<8..15>	i, d, n, s, sf, x, sx, sfx	NZVC
1100 raaa	LDWR	Load word r from memory	i, d, n, s, sf, x, sx, sfx	NZ
1101 raaa	LDBR	Load byte r<8..15> from memory	i, d, n, s, sf, x, sx, sfx	NZ
1110 raaa	STWR	Store word r to memory	d, n, s, sf, x, sx, sfx	
1111 raaa	STBR	Store byte r<8..15> to memory	d, n, s, sf, x, sx, sfx	

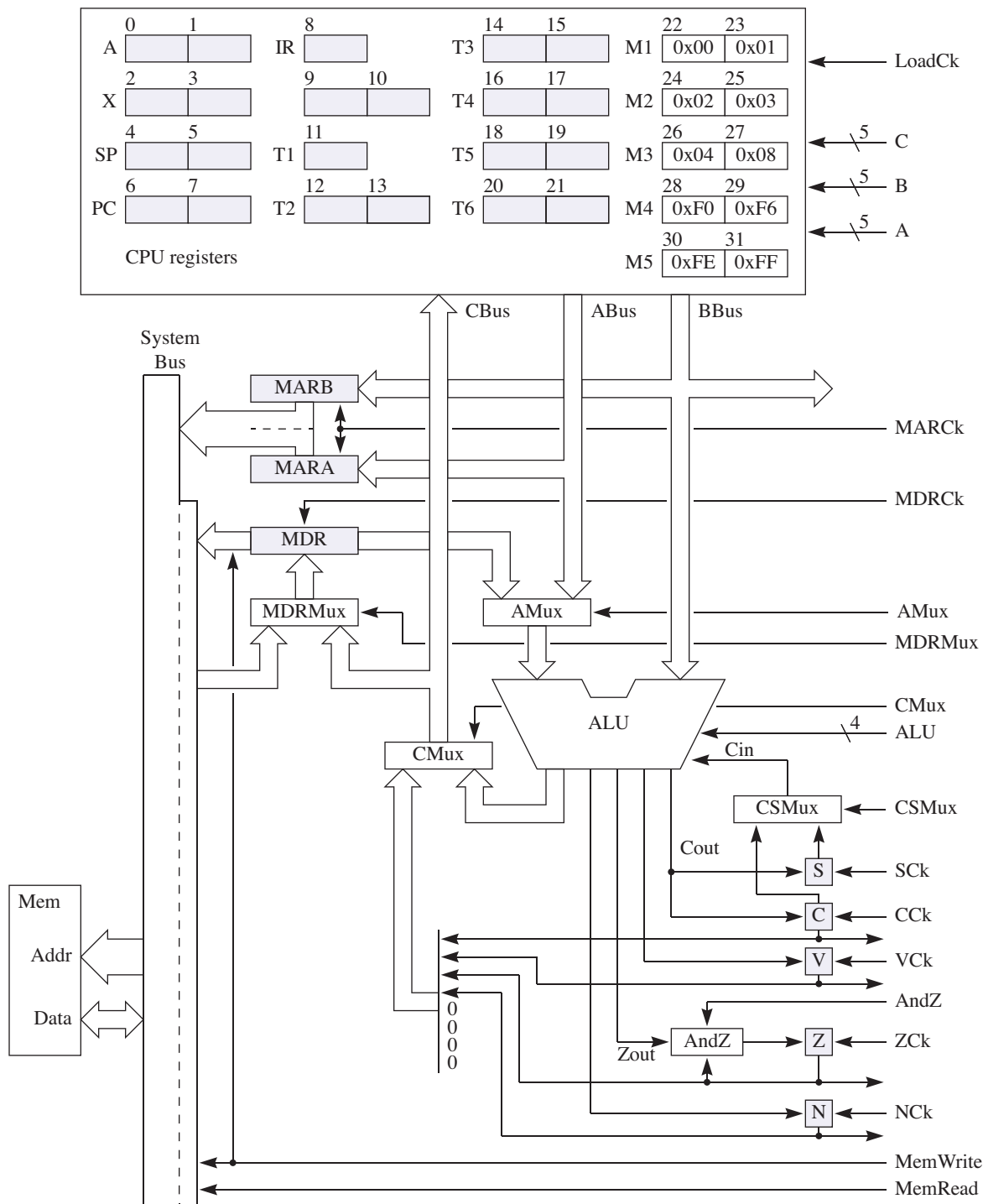
Here is the RTL specification of the Pep/9 instruction set.

Instruction	Register transfer language specification
STOP	Stop execution
RET	$PC \leftarrow \text{Mem}[SP] ; SP \leftarrow SP + 2$
RETR	$NZVC \leftarrow \text{Mem}[SP] \langle 4..7 \rangle ; A \leftarrow \text{Mem}[SP + 1] ; X \leftarrow \text{Mem}[SP + 3] ; PC \leftarrow \text{Mem}[SP + 5] ; SP \leftarrow \text{Mem}[SP + 7]$
MOVSPA	$A \leftarrow SP$
MOVFLGA	$A \langle 8..11 \rangle \leftarrow 0 , A \langle 12..15 \rangle \leftarrow NZVC$
MOVAFLG	$NZVC \leftarrow A \langle 12..15 \rangle$
NOTr	$r \leftarrow \neg r ; N \leftarrow r < 0 , Z \leftarrow r = 0$
NEGr	$r \leftarrow -r ; N \leftarrow r < 0 , Z \leftarrow r = 0 , V \leftarrow \{overflow\}$
ASLr	$C \leftarrow r \langle 0 \rangle , r \langle 0..14 \rangle \leftarrow r \langle 1..15 \rangle , r \langle 15 \rangle \leftarrow 0 ; N \leftarrow r < 0 , Z \leftarrow r = 0 , V \leftarrow \{overflow\}$
ASRr	$C \leftarrow r \langle 15 \rangle , r \langle 1..15 \rangle \leftarrow r \langle 0..14 \rangle ; N \leftarrow r < 0 , Z \leftarrow r = 0$
ROLr	$C \leftarrow r \langle 0 \rangle , r \langle 0..14 \rangle \leftarrow r \langle 1..15 \rangle , r \langle 15 \rangle \leftarrow C$
RORr	$C \leftarrow r \langle 15 \rangle , r \langle 1..15 \rangle \leftarrow r \langle 0..14 \rangle , r \langle 0 \rangle \leftarrow C$
BR	$PC \leftarrow \text{Oprnd}$
BRLE	$N = 1 \vee Z = 1 \Rightarrow PC \leftarrow \text{Oprnd}$
BRLT	$N = 1 \Rightarrow PC \leftarrow \text{Oprnd}$
BREQ	$Z = 1 \Rightarrow PC \leftarrow \text{Oprnd}$
BRNE	$Z = 0 \Rightarrow PC \leftarrow \text{Oprnd}$
BRGE	$N = 0 \Rightarrow PC \leftarrow \text{Oprnd}$
BRGT	$N = 0 \wedge Z = 0 \Rightarrow PC \leftarrow \text{Oprnd}$
BRV	$V = 1 \Rightarrow PC \leftarrow \text{Oprnd}$
BRC	$C = 1 \Rightarrow PC \leftarrow \text{Oprnd}$
CALL	$SP \leftarrow SP - 2 ; \text{Mem}[SP] \leftarrow PC ; PC \leftarrow \text{Oprnd}$
NOPn	Trap: Unary no operation
NOP	Trap: Nonunary no operation
DECI	Trap: $\text{Oprnd} \leftarrow \{decimal\ input\}$
DECO	Trap: $\{decimal\ output\} \leftarrow \text{Oprnd}$
HEXO	Trap: $\{hexadecimal\ output\} \leftarrow \text{Oprnd}$
STRO	Trap: $\{string\ output\} \leftarrow \text{Oprnd}$
ADDSP	$SP \leftarrow SP + \text{Oprnd}$
SUBSP	$SP \leftarrow SP - \text{Oprnd}$
ADDr	$r \leftarrow r + \text{Oprnd} ; N \leftarrow r < 0 , Z \leftarrow r = 0 , V \leftarrow \{overflow\} , C \leftarrow \{carry\}$
SUBr	$r \leftarrow r - \text{Oprnd} ; N \leftarrow r < 0 , Z \leftarrow r = 0 , V \leftarrow \{overflow\} , C \leftarrow \{carry\}$
ANDr	$r \leftarrow r \wedge \text{Oprnd} ; N \leftarrow r < 0 , Z \leftarrow r = 0$
ORr	$r \leftarrow r \vee \text{Oprnd} ; N \leftarrow r < 0 , Z \leftarrow r = 0$
CPWr	$T \leftarrow r - \text{Oprnd} ; N \leftarrow T < 0 , Z \leftarrow T = 0 , V \leftarrow \{overflow\} , C \leftarrow \{carry\} ; N \leftarrow N \oplus V$
CPBr	$T \leftarrow r \langle 8..15 \rangle - \text{byte Oprnd} ; N \leftarrow T < 0 , Z \leftarrow T = 0 , V \leftarrow 0 , C \leftarrow 0$
LDWr	$r \leftarrow \text{Oprnd} ; N \leftarrow r < 0 , Z \leftarrow r = 0$
LDBr	$r \langle 8..15 \rangle \leftarrow \text{byte Oprnd} ; N \leftarrow 0 , Z \leftarrow r \langle 8..15 \rangle = 0$
STWr	$\text{Oprnd} \leftarrow r$
STBr	$\text{byte Oprnd} \leftarrow r \langle 8..15 \rangle$
Trap	$T \leftarrow \text{Mem}[\text{FFF6}] ; \text{Mem}[T - 1] \leftarrow \text{IR} \langle 0..7 \rangle ; \text{Mem}[T - 3] \leftarrow SP ; \text{Mem}[T - 5] \leftarrow PC ; \text{Mem}[T - 7] \leftarrow X ; \text{Mem}[T - 9] \leftarrow A ; \text{Mem}[T - 10] \langle 4..7 \rangle \leftarrow NZVC ; SP \leftarrow T - 10 ; PC \leftarrow \text{Mem}[\text{FFFE}]$

Here is the memory map of the Pep/9 system. The shaded portion is ROM. Compared to the Pep/8 memory map, this map has an input port at address FC15 and an output port at address FC16. Corresponding to the I/O ports are two additional machine vectors at addresses FFF8 and FFFA.



Here is the data section of the Pep/9 CPU. Compared to the Pep/8 data section, Pep/9 has two additional components — a shadow carry bit, denoted S in the figure below, and an additional multiplexer with its associated control line CSMux. The shadow carry bit is not visible at the ISA level and is used for internal address calculations in the microcode. This design solves a major headache present in Pep/8, which requires the saving and restoration of the C bit when an internal address addition would wipe it out. A step towards a more realistic model is the requirement of three consecutive MemRead/MemWrite assertions for memory access as opposed to two with Pep/8.



Here is the data section of the Pep/9 CPU with the two-byte data bus. The fifth edition of *Computer Systems* drops the discussion of the MAR Incrementer in favor of a more extensive discussion of increasing the data bus width to improve performance. The material is improved by incorporating it into the Pep9CPU software. Students can toggle between the two models, with and without the wider data bus, test their solutions with the software, and use the UnitPre and UnitPost tests in the Help system.

