

Pep/10

J. Stanley Warford

June 5, 2019

Here are the differences between Pep/10 and Pep/9 along with a rationale for each change.

1. STOP replaced by RET

The STOP instruction is no longer in the instruction set. Instead, the operating system now calls the C `main()` function with the system return value preset to 0. The translation more closely matches the terminating C statement

```
return 0;
```

The symbolic debugger of the Pep/10 IDE now shows the run-time stack from the OS call with two cells – `retAddr` and `retVal`. If students terminate their programs with RET the return value will be 0 because that is the preset return value, and control is returned to the simulator the same way a STOP instruction does in Pep/9. However, if they modify the value before the return, the OS issues an error message with an echo of the error number.

The operating system has a new dispatcher component as the interface between the OS and the application. This interface is more realistic of the way C works and reinforces the concept that the operating system calls the application, and the application returns control to the operating system.

2. Memory-mapped shutdown port

Pep/9 introduced the concept of memory-mapped I/O ports. In another step toward hardware realism, and to have a mechanism for terminating a simulation, Pep/10 has a memory-mapped shutdown port. If any value at all is written to the port the simulation is terminated and control is returned to the IDE.

Students first learn how to program in machine language at the ISA3 level without the assistance of the operating system. They learn how to store a byte to the output port with direct addressing to output an ASCII character. In Pep/10, they simply store a byte to the shutdown port with direct addressing to terminate their programs.

There are two benefits to this feature. First, students do not need to learn a new STOP instruction to terminate their machine language programs. But more importantly, they learn the utility of memory-mapped device registers with this rudimentary example.

3. System calls

Blah, blah ...

Of all the instructions in the Pep/9 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/10. In the new instruction set, there are no native input and output instructions. Instead, the Pep/9 instruction

```
CHARI alpha,ad
```

is replaced by the Pep/10 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/9 instruction

```
CHARO beta,ad
```

is replaced by the Pep/10 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/10 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/9 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/9. 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.

4. New native instruction CPBr

In Pep/9, byte quantities must be compared with CPr, 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/9 load byte instruction LDBYTEr is

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

The RTL specification of the Pep/10 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.

5. Improved mnemonics

Pep/10 renames the mnemonics for the compare, load, and store instructions as shown below.

Instruction	Pep/10	Pep/9
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/10 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”.

6. New trap instruction HEXO

Pep/10 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/10. It outputs a word as four hexadecimal characters.

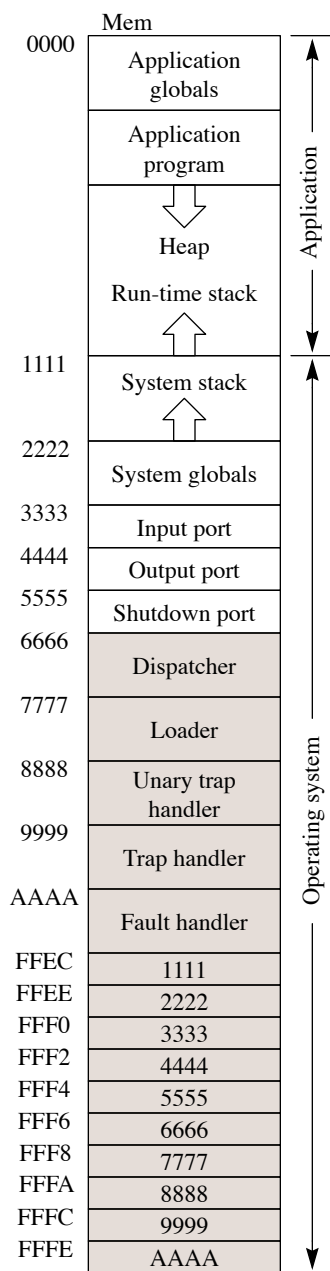
7. Addressing mode nomenclature

Pep/10 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.

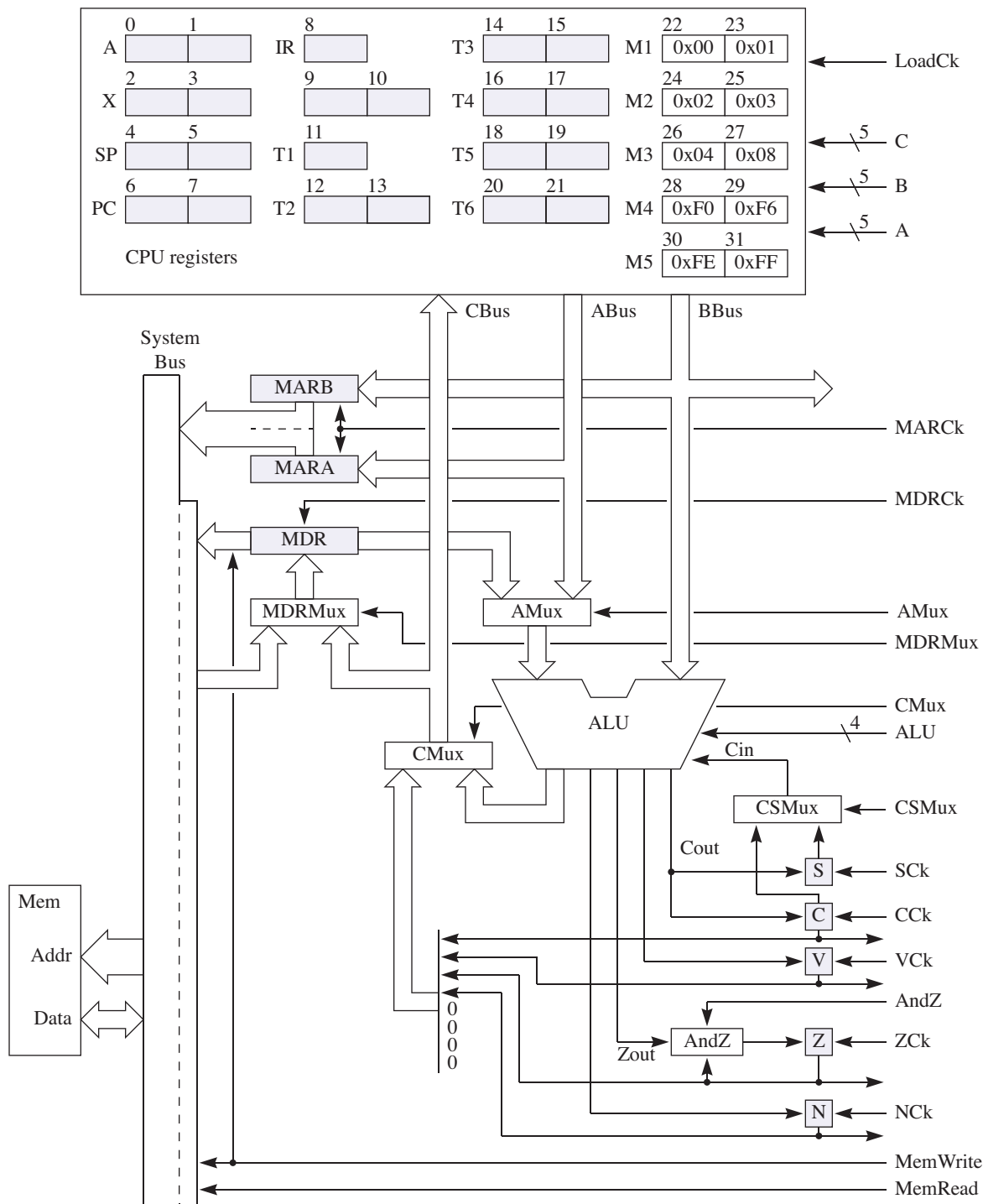
Instruction Specifier	Mnemonic	Instruction	Addressing Mode	Status Bits
0000 0000	RET	Return from CALL	U	
0000 0001	RETSY	Return from system CALL	U	
0000 0010	MOVSPA	Move SP to A	U	
0000 0011	MOVASP	Move A to SP	U	
0000 0100	MOVFLGA	Move NZVC flags to A(12..15)	U	
0000 0101	MOVAFLG	Move A(12..15) to NZVC flags	U	
0000 0110	MOVTPC	Move T to PC	U	
0000 0111	NOP	No operation	U	
0000 1000	USYCALL	Unary system call	U	
0001 000r	NOTr	Bitwise invert r	U	NZ
0001 001r	NEGr	Negate r	U	NZV
0001 010r	ASLr	Arithmetic shift left r	U	NZVC
0001 011r	ASRr	Arithmetic shift right r	U	NZC
0001 100r	ROLr	Rotate left r	U	C
0001 101r	RORr	Rotate right r	U	C
0001 110a	BR	Branch unconditional	i, x	
0001 111a	BRLE	Branch if less than or equal to	i, x	
0010 000a	BRLT	Branch if less than	i, x	
0010 001a	BREQ	Branch if equal to	i, x	
0010 010a	BRNE	Branch if not equal to	i, x	
0010 011a	BRGE	Branch if greater than or equal to	i, x	
0010 100a	BRGT	Branch if greater than	i, x	
0010 101a	BRV	Branch if V	i, x	
0010 110a	BRC	Branch if C	i, x	
0010 111a	CALL	Call subroutine	i, x	
0011 0aaa	SYCALL	System call	i, d, n, s, sf, x, sx, sfx	
0011 1aaa	LDWT	Load word T from memory	i, d, n, s, sf, x, sx, sfx	
0100 raaa	LDWr	Load word r from memory	i, d, n, s, sf, x, sx, sfx	NZ
0101 raaa	LDBr	Load byte r(8..15) from memory	i, d, n, s, sf, x, sx, sfx	NZ
0110 raaa	STWr	Store word r to memory	d, n, s, sf, x, sx, sfx	
0111 raaa	STBr	Store byte r(8..15) to memory	d, n, s, sf, x, sx, sfx	
1000 raaa	CPWr	Compare word to r	i, d, n, s, sf, x, sx, sfx	NZVC
1001 raaa	CPBr	Compare byte to r(8..15)	i, d, n, s, sf, x, sx, sfx	NZVC
1010 raaa	ADDr	Add to r	i, d, n, s, sf, x, sx, sfx	NZVC
1011 raaa	SUBr	Subtract from r	i, d, n, s, sf, x, sx, sfx	NZVC
1100 raaa	ANDr	Bitwise AND to r	i, d, n, s, sf, x, sx, sfx	NZ
1101 raaa	ORr	Bitwise OR to r	i, d, n, s, sf, x, sx, sfx	NZ
1110 raaa	XORr	Bitwise XOR to r	i, d, n, s, sf, x, sx, sfx	NZ
1111 0aaa	ADDSP	Add to SP	i, d, n, s, sf, x, sx, sfx	NZVC
1111 1aaa	SUBSP	Subtract from SP	i, d, n, s, sf, x, sx, sfx	NZVC

Instruction	Register transfer language specification
RET	$PC \leftarrow \text{Mem}[SP]; SP \leftarrow SP + 2$
RETSY	$NZVC \leftarrow \text{Mem}[SP]\langle 4..7 \rangle; A \leftarrow \text{Mem}[SP + 1]; X \leftarrow \text{Mem}[SP + 3]; PC \leftarrow \text{Mem}[SP + 5]; SP \leftarrow \text{Mem}[SP + 7]$
MOVSPA	$A \leftarrow SP$
MOVASP	$SP \leftarrow A$
MOVFLGA	$A\langle 8..11 \rangle \leftarrow 0, A\langle 12..15 \rangle \leftarrow NZVC$
MOVAFLG	$NZVC \leftarrow A\langle 12..15 \rangle$
MOVTPC	$PC \leftarrow T$
NOP	<i>{No operation}</i>
USYCALL	$Y \leftarrow \text{Mem}[\text{FFF6}]; \text{Mem}[Y - 1] \leftarrow \text{IR}\langle 0..7 \rangle; \text{Mem}[Y - 3] \leftarrow SP; \text{Mem}[Y - 5] \leftarrow PC; \text{Mem}[Y - 7] \leftarrow Y;$ $\text{Mem}[Y - 9] \leftarrow A; \text{Mem}[Y - 10]\langle 4..7 \rangle \leftarrow NZVC; SP \leftarrow Y - 10; PC \leftarrow \text{Mem}[\text{FFFA}]$
LDWT	$T \leftarrow \text{Oprnd}$
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	$SP \leftarrow SP - 2; \text{Mem}[SP] \leftarrow PC; PC \leftarrow \text{Oprnd}$
SYCALL	$Y \leftarrow \text{Mem}[\text{FFF6}]; \text{Mem}[Y - 1] \leftarrow \text{IR}\langle 0..7 \rangle; \text{Mem}[Y - 3] \leftarrow SP; \text{Mem}[Y - 5] \leftarrow PC; \text{Mem}[Y - 7] \leftarrow Y;$ $\text{Mem}[Y - 9] \leftarrow A; \text{Mem}[Y - 10]\langle 4..7 \rangle \leftarrow NZVC; SP \leftarrow Y - 10; PC \leftarrow \text{Mem}[\text{FFFC}]$
LDWT	$T \leftarrow \text{Oprnd}$
LDWrr	$r \leftarrow \text{Oprnd}; N \leftarrow r < 0, Z \leftarrow r = 0$
LDBrr	$r\langle 8..15 \rangle \leftarrow \text{byte Oprnd}; N \leftarrow 0, Z \leftarrow r\langle 8..15 \rangle = 0$
STWrr	$\text{Oprnd} \leftarrow r$
STBrr	$\text{byte Oprnd} \leftarrow r\langle 8..15 \rangle$
CPWrr	$Y \leftarrow r - \text{Oprnd}; N \leftarrow Y < 0, Z \leftarrow Y = 0, V \leftarrow \{overflow\}, C \leftarrow \{carry\}; N \leftarrow N \oplus V$
CPBrr	$Y \leftarrow r\langle 8..15 \rangle - \text{byte Oprnd}; N \leftarrow Y < 0, Z \leftarrow Y = 0, V \leftarrow 0, C \leftarrow 0$
ADDrr	$r \leftarrow r + \text{Oprnd}; N \leftarrow r < 0, Z \leftarrow r = 0, V \leftarrow \{overflow\}, C \leftarrow \{carry\}$
SUBrr	$r \leftarrow r - \text{Oprnd}; N \leftarrow r < 0, Z \leftarrow r = 0, V \leftarrow \{overflow\}, C \leftarrow \{carry\}$
ANDrr	$r \leftarrow r \wedge \text{Oprnd}; N \leftarrow r < 0, Z \leftarrow r = 0$
ORrr	$r \leftarrow r \vee \text{Oprnd}; N \leftarrow r < 0, Z \leftarrow r = 0$
XORrr	$r \leftarrow r \oplus \text{Oprnd}; N \leftarrow r < 0, Z \leftarrow r = 0$
ADDSP	$SP \leftarrow SP + \text{Oprnd}$
SUBSP	$SP \leftarrow SP - \text{Oprnd}$

Here is the memory map of the Pep/10 system. The shaded portion is ROM. Compared to the Pep/9 memory map, this map has several additional components – a shutdown port at address 5555, a dispatcher at address 6666, a unary trap handler at address 8888 and a fault handler at address AAAA. Pep/9 has six machine vectors from FFF4 to FFFE. Because of the new components, Pep/10 has ten machine vectors at addresses FFEC to FFFE.



Here is the data section of the Pep/10 CPU. Compared to the Pep/9 data section, Pep/10 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/9, 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/9.



Here is the data section of the Pep/10 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.

