Pep/10

J. Stanley Warford

June 4, 2019

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

1. Change rtl specification for SUBr and CPWr to negate operand into temporary in separate step.

2. `RETn` replaced by `RET`

   In Pep/9, 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/10 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.

3. Memory-mapped I/O

   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

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

The RTL specification of the Pep/10 load byte instruction, now named `LDBr`, is

$r\langle 8..15\rangle \leftarrow$ 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`	*Not available*
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.

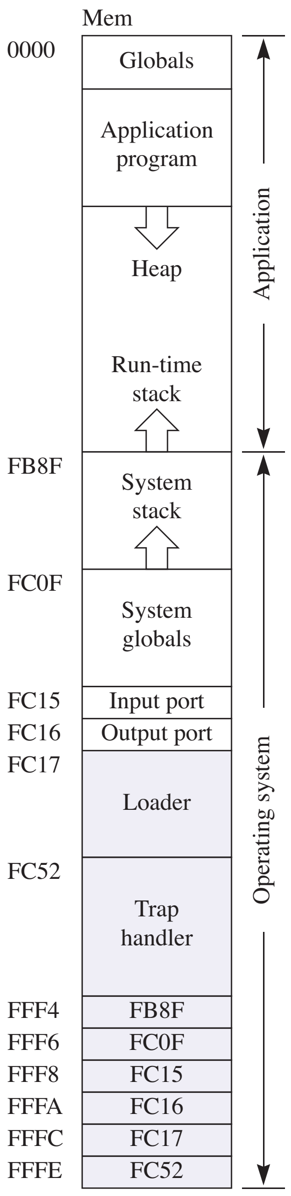
7. Addressing mode nomenclature

   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.

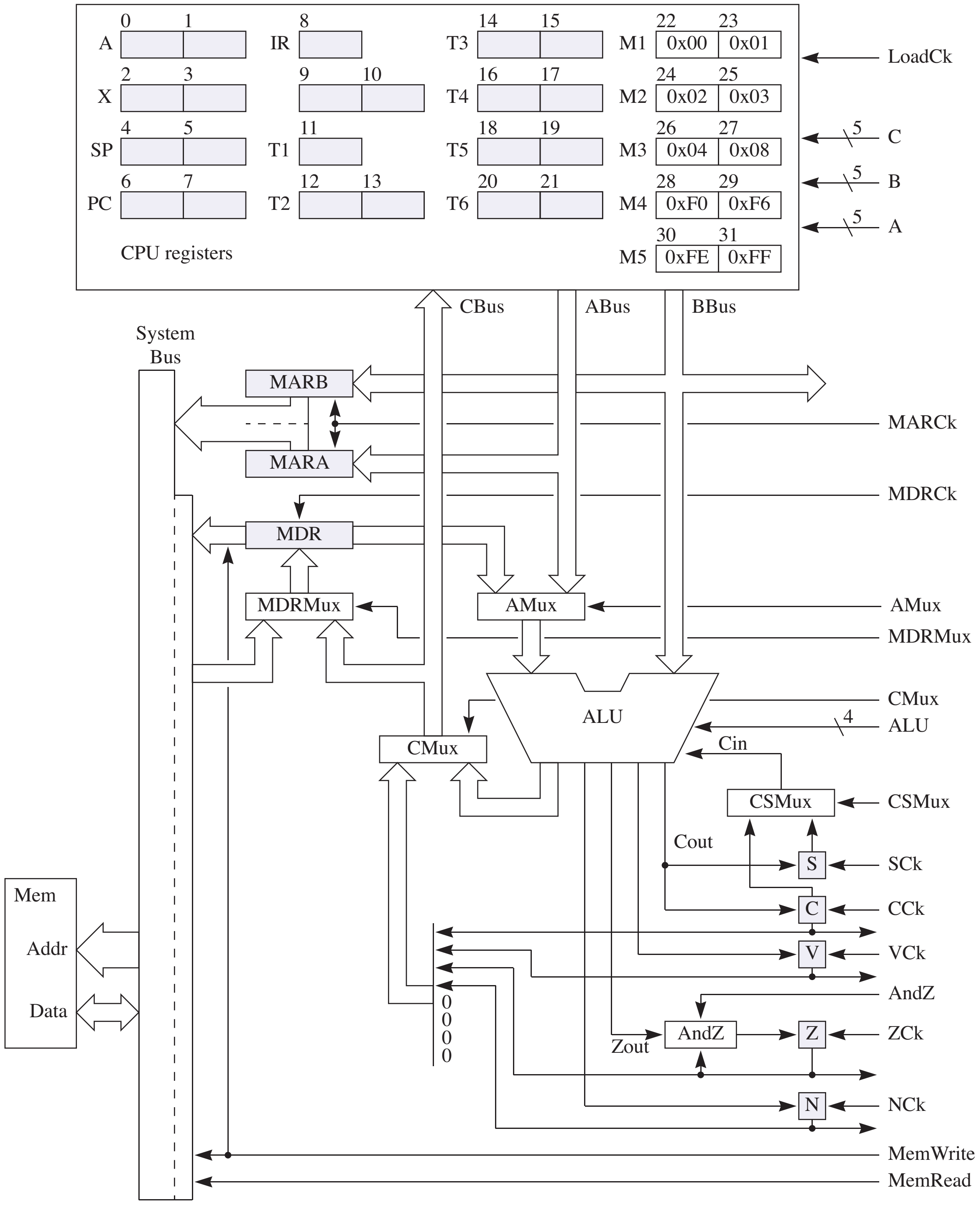
Instruction Specifier	Mnemonic	Instruction	Addressing Mode	Status Bits
0000 0000	`RET`	Return from `CALL`	U	
0000 0001	`RETSY`	Return from system `CALL`	U	
0000 0010	`MOVSPA`	Move SP to A	U	
0000 0011	`MOVASP`	Move A to SP	U	
0000 0100	`MOVFLGA`	Move NZVC flags to A⟨12..15⟩	U	
0000 0101	`MOVAFLG`	Move 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`	$\mathrm{PC} \leftarrow \mathrm{Mem[SP]}$; $\mathrm{SP} \leftarrow \mathrm{SP}+2$
`RETSY`	$\mathrm{NZVC} \leftarrow \mathrm{Mem[SP]}\langle 4..7\rangle$; $\mathrm{A} \leftarrow \mathrm{Mem[SP+1]}$; $\mathrm{X} \leftarrow \mathrm{Mem[SP+3]}$; $\mathrm{PC} \leftarrow \mathrm{Mem[SP+5]}$; $\mathrm{SP} \leftarrow \mathrm{Mem[SP+7]}$
`MOVSPA`	$\mathrm{A} \leftarrow \mathrm{SP}$
`MOVASP`	$\mathrm{SP} \leftarrow \mathrm{A}$
`MOVFLGA`	$\mathrm{A}\langle 8..11\rangle \leftarrow 0$, $\mathrm{A}\langle 12..15\rangle \leftarrow \mathrm{NZVC}$
`MOVAFLG`	$\mathrm{NZVC} \leftarrow \mathrm{A}\langle 12..15\rangle$
`MOVTPC`	$\mathrm{PC} \leftarrow \mathrm{T}$
`NOP`	{*No operation*}
`USYCALL`	
`NOTr`	$\mathrm{r} \leftarrow \neg\mathrm{r}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$
`NEGr`	$\mathrm{r} \leftarrow -\mathrm{r}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$, $\mathrm{V} \leftarrow \{\mathit{overflow}\}$
`ASLr`	$\mathrm{C} \leftarrow \mathrm{r}\langle 0\rangle$, $\mathrm{r}\langle 0..14\rangle \leftarrow \mathrm{r}\langle 1..15\rangle$, $\mathrm{r}\langle 15\rangle \leftarrow 0$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$, $\mathrm{V} \leftarrow \{\mathit{overflow}\}$
`ASRr`	$\mathrm{C} \leftarrow \mathrm{r}\langle 15\rangle$, $\mathrm{r}\langle 1..15\rangle \leftarrow \mathrm{r}\langle 0..14\rangle$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$
`ROLr`	$\mathrm{C} \leftarrow \mathrm{r}\langle 0\rangle$, $\mathrm{r}\langle 0..14\rangle \leftarrow \mathrm{r}\langle 1..15\rangle$, $r\langle 15\rangle \leftarrow \mathrm{C}$
`RORr`	$\mathrm{C} \leftarrow \mathrm{r}\langle 15\rangle$, $\mathrm{r}\langle 1..15\rangle \leftarrow \mathrm{r}\langle 0..14\rangle$, $r\langle 0\rangle \leftarrow \mathrm{C}$
`BR`	$\mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BRLE`	$\mathrm{N} = 1 \vee \mathrm{Z} = 1 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BRLT`	$\mathrm{N} = 1 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BREQ`	$\mathrm{Z} = 1 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BRNE`	$\mathrm{Z} = 0 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BRGE`	$\mathrm{N} = 0 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BRGT`	$\mathrm{N} = 0 \wedge \mathrm{Z} = 0 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BRV`	$\mathrm{V} = 1 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`BRC`	$\mathrm{C} = 1 \Rightarrow \mathrm{PC} \leftarrow \mathrm{Oprnd}$
`CALL`	$\mathrm{SP} \leftarrow \mathrm{SP}-2$; $\mathrm{Mem[SP]} \leftarrow \mathrm{PC}$; $\mathrm{PC} \leftarrow \mathrm{Oprnd}$
`SYCALL`	$\mathrm{X} \leftarrow \mathrm{Mem[FFF6]}$; $\mathrm{Mem}[\mathrm{X}-1] \leftarrow \mathrm{IR}\langle 0..7\rangle$; $\mathrm{Mem}[\mathrm{X}-3] \leftarrow \mathrm{SP}$; $\mathrm{Mem}[\mathrm{X}-5] \leftarrow \mathrm{PC}$; $\mathrm{Mem}[\mathrm{X}-7] \leftarrow \mathrm{X}$; $\mathrm{Mem}[\mathrm{X}-9] \leftarrow \mathrm{A}$; $\mathrm{Mem}[\mathrm{X}-10]\langle 4..7\rangle \leftarrow \mathrm{NZVC}$; $\mathrm{SP} \leftarrow \mathrm{X}-10$; $\mathrm{PC} \leftarrow \mathrm{Mem[FFFE]}$
`LDWT`	$\mathrm{T} \leftarrow \mathrm{Oprnd}$
`LDWr`	$\mathrm{r} \leftarrow \mathrm{Oprnd}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$
`LDBr`	$\mathrm{r}\langle 8..15\rangle \leftarrow$ byte Oprnd ; $\mathrm{N} \leftarrow 0$, $\mathrm{Z} \leftarrow \mathrm{r}\langle 8..15\rangle = 0$
`STWr`	$\mathrm{Oprnd} \leftarrow \mathrm{r}$
`STBr`	byte $\mathrm{Oprnd} \leftarrow \mathrm{r}\langle 8..15\rangle$
`CPWr`	$\mathrm{X} \leftarrow \mathrm{r} - \mathrm{Oprnd}$; $\mathrm{N} \leftarrow \mathrm{X} < 0$, $\mathrm{Z} \leftarrow \mathrm{X} = 0$, $\mathrm{V} \leftarrow \{\mathit{overflow}\}$, $\mathrm{C} \leftarrow \{\mathit{carry}\}$; $\mathrm{N} \leftarrow \mathrm{N} \oplus \mathrm{V}$
`CPBr`	$\mathrm{X} \leftarrow \mathrm{r}\langle 8..15\rangle -$ byte Oprnd ; $\mathrm{N} \leftarrow \mathrm{X} < 0$, $\mathrm{Z} \leftarrow \mathrm{X} = 0$, $\mathrm{V} \leftarrow 0$, $\mathrm{C} \leftarrow 0$
`ADDr`	$\mathrm{r} \leftarrow \mathrm{r} + \mathrm{Oprnd}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$, $\mathrm{V} \leftarrow \{\mathit{overflow}\}$, $\mathrm{C} \leftarrow \{\mathit{carry}\}$
`SUBr`	$\mathrm{r} \leftarrow \mathrm{r} - \mathrm{Oprnd}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$, $\mathrm{V} \leftarrow \{\mathit{overflow}\}$, $\mathrm{C} \leftarrow \{\mathit{carry}\}$
`ANDr`	$\mathrm{r} \leftarrow \mathrm{r} \wedge \mathrm{Oprnd}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$
`ORr`	$\mathrm{r} \leftarrow \mathrm{r} \vee \mathrm{Oprnd}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$
`XORr`	$\mathrm{r} \leftarrow \mathrm{r} \oplus \mathrm{Oprnd}$; $\mathrm{N} \leftarrow \mathrm{r} < 0$, $\mathrm{Z} \leftarrow \mathrm{r} = 0$
`ADDSP`	$\mathrm{SP} \leftarrow \mathrm{SP} + \mathrm{Oprnd}$
`SUBSP`	$\mathrm{SP} \leftarrow \mathrm{SP} - \mathrm{Oprnd}$

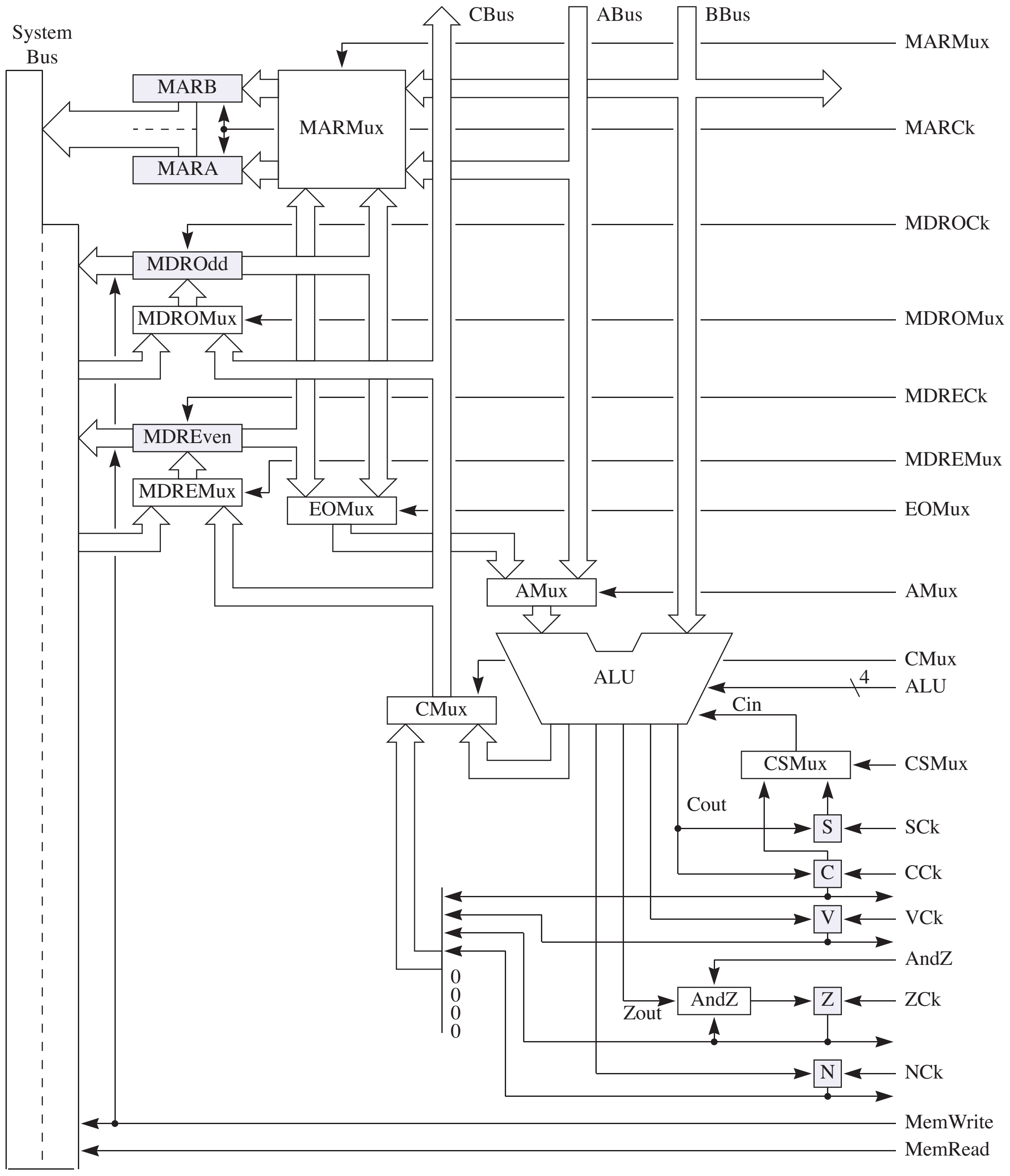
Here is the memory map of the Pep/10 system. The shaded portion is ROM. Compared to the Pep/9 memory map, this map has an input port at address FC15 and an output port at address FC16. Corresponding to the I/O ports are two additional machine vectors at addresses FFF8 and FFFA.



Here is the data section of the Pep/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.


System Bus
CBus
ABus
BBus
MARMux
MARB
MARA
MARCk
MDROCk
MDROdd
MDROMux
MDRECk
MDREven
MDREMux
EOMux
AMux
ALU
CMux
4
Cin
CSMux
Cout
S
SCk
C
CCk
V
VCk
AndZ
Zout
Z
ZCk
N
NCk
0
0
0
0
MemWrite
MemRead