# Pep/10

J. Stanley Warford
Matthew McRaven

January 16, 2021

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 reenforces 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. Trap instructions replaced by system calls

   Pep/9 has five trap instructions – `NOP`, `DECI`, `DECO`, `HEXO`, and `STRO`. Pep/10 replaces them with two system calls – `SCALL` for system call and `USCALL` for unary system call. For example, `DECI` is no longer an instruction mnemonic as it is in Pep/9, but a symbol exported from the operating system. This decimal input instruction in Pep/9

   ```
   DECI   num,d
   ```

   becomes the following system call in Pep/10

   ```
   LDWT   DECI,i
   SCALL  num,d
   ```

   Pep/10 has a new Trap register visible at level ISA3 used by the system call instructions. `LDWT` is the load word trap instruction, which loads the entry point address of the `DECI` code in the operating system.

   In Pep/9, the use of the trap instructions so closely parallels the use of the native ISA instructions that beginning students frequently do not even realize they are system calls. Pep/10 has the pedagogic advantage of making system calls explicit, and is more realistic. From a system design perspective, the trap instructions no longer occupy the opcode space which opens the possibility for new instructions. The new design also does not constrain the number or variety of system calls.

4. Assembler macro facility

   In yet another step toward industry standard practice, Pep/10 introduces assembler macro expansions. The IDE provides cononical standard macros for all the system calls. With the supplied macros, the above `DECI` example becomes simply

   ```
   @DECI  num,d
   ```

where `@DECI` is now the macro name. The generated program listing shows the macro source instruction and its expansion. The convenient `CHARI` and `CHARO` instructions from Pep/8 and earlier are now back as macros `@CHARI` and `@CHARO`. This is a pedagogical improvement over Pep/9, because now character I/O is programmed exactly like decimal I/O. With dynamic allocation, students no longer need to copy/paste the code for `malloc()` at the end of their source because `@MALLOC` is a supplied macro.

It is possible for students to write their own macros with the Pep/10 IDE, which contains documentation for how to do so. However, writing macros is outside the scope of this text.

5. Easier modification of OS

   In Pep/9, to write a new trap instruction the IDE requires you to redefine one of the existing mnemonics. Now that trap instructions are replaced with system calls there is no longer such a requirement. The Pep/10 assembler uses the new `.SCALL` and `.USCALL` directives to automatically create system call macros. Operating system programmers may use the `.EXPORT` directive to control which symbols are available from application code. By combining these facilities, students can write any number of system calls unconstrained by the opcode space.

   The full declaration of a system call in the operating system is as follows:

```
         .EXPORT mymacro
         .USCALL mymacro
mymacro: RET
```

6. Explicitly declared IO ports

   Pep/10 introduces two new directives: `.INPUT` and `.OUTPUT`. These directives indicate to the simulation and student that associated symbols correspond to user-accessible devices. Operating system programmers will receive warnings from the Pep/10 assembler if they have disconnected IO devices, preventing silent IO failures caused by misspelled symbols. An IO directive does not automatically mark the symbol as `EXPORT`'ed, maintaining parity with system call declarations.

   The full declaration for a memory-mapped IO port is as follows:

```
      .EXPORT port
      .OUTPUT port
port: .BLOCK  1
```

7. New disk input port

   In another bid towards a more realistic system model, the loader now takes its input from a disk input port.

8. New instruction `XORr`

   Deletion of the trap instructions from Pep/9 opened up the opcode space for new instructions. Finally, the exclusive OR instruction `XORr` is a native ISA instruction.

9. Improved instruction set

   The instruction set is now more representative of real ISA instruction sets. Neither the `STOP` instruction nor the specialized trap mnemonics of Pep/9 are in actual ISA sets. System calls are also typical. Both Pep/9 and Pep/10 have 40 ISA instructions, but the Pep/10 set is more regular. The opcode space is cleaner, with all the unary instructions followed by all the branch instructions with two addressing modes followed by the remaining instructions with eight addressing modes.

10. Modified CPU

    At the ISA level, the only difference in the CPU is the new 16-bit Trap register, which is necessary to make system calls. At the LG1 level, the Pep/10 data section for the one-byte bus is identical to that of Pep/9, again with the only difference being the labeling of the Trap register. The two-byte bus model is deleted altogether in favor of expanded coverage of an industry standard RISC chip.

11. New RISC-V microarchitecture

    The MIPS processor in previous editions is replaced by the open-source hardware standard RISC-V, which is presented throughout the text. The sidebars that describe the x86 architecture in the previous edition now describe RISC-V.

The previous edition contains microcode programming problems for the one-byte bus and two-byte bus versions of the Pep/9 CPU data section. MIPS assembly language exercises are limited to paper. This edition replaces the MIPS paper exercises with RISC-V assembly language programming problems. Students implement components of the Pep/10 computer at the Mc2 level with microcode as before. Now they also implement components of the Pep/10 computer at the ISA level with RISC-V assembly language code.

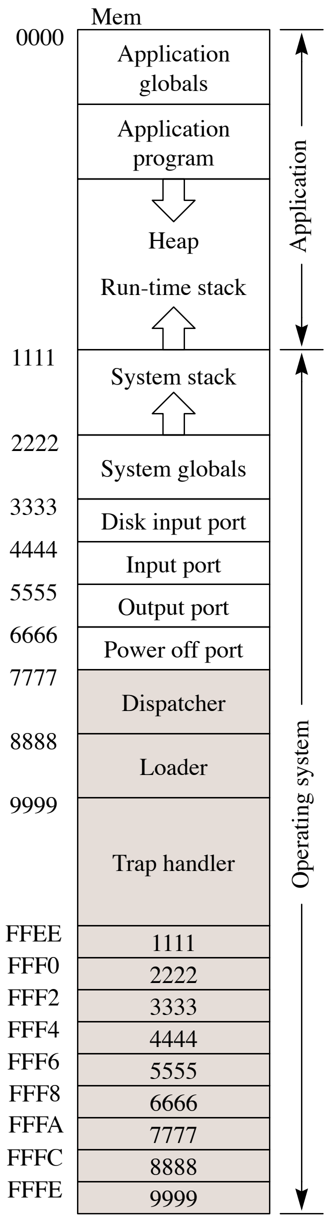
12. New software support

    The suite of software for Pep/10 includes an assembler/simulator at the ISA level and a micro-assembler and simulator at the Mc2 level as before. The micro-assembler has a unit pre- and post-condition facility for students to test their programs. The suite now includes a RISC-V assembler/simulator with the same pre- and post-condition facility. The application maps the Pep/10 ISA registers onto a subset of the RISC-V registers for students to simulate Pep/10 at the ISA level. They experience the difference between CISC and RISC using RISC-V as a form of microcode to simulate Pep/10.

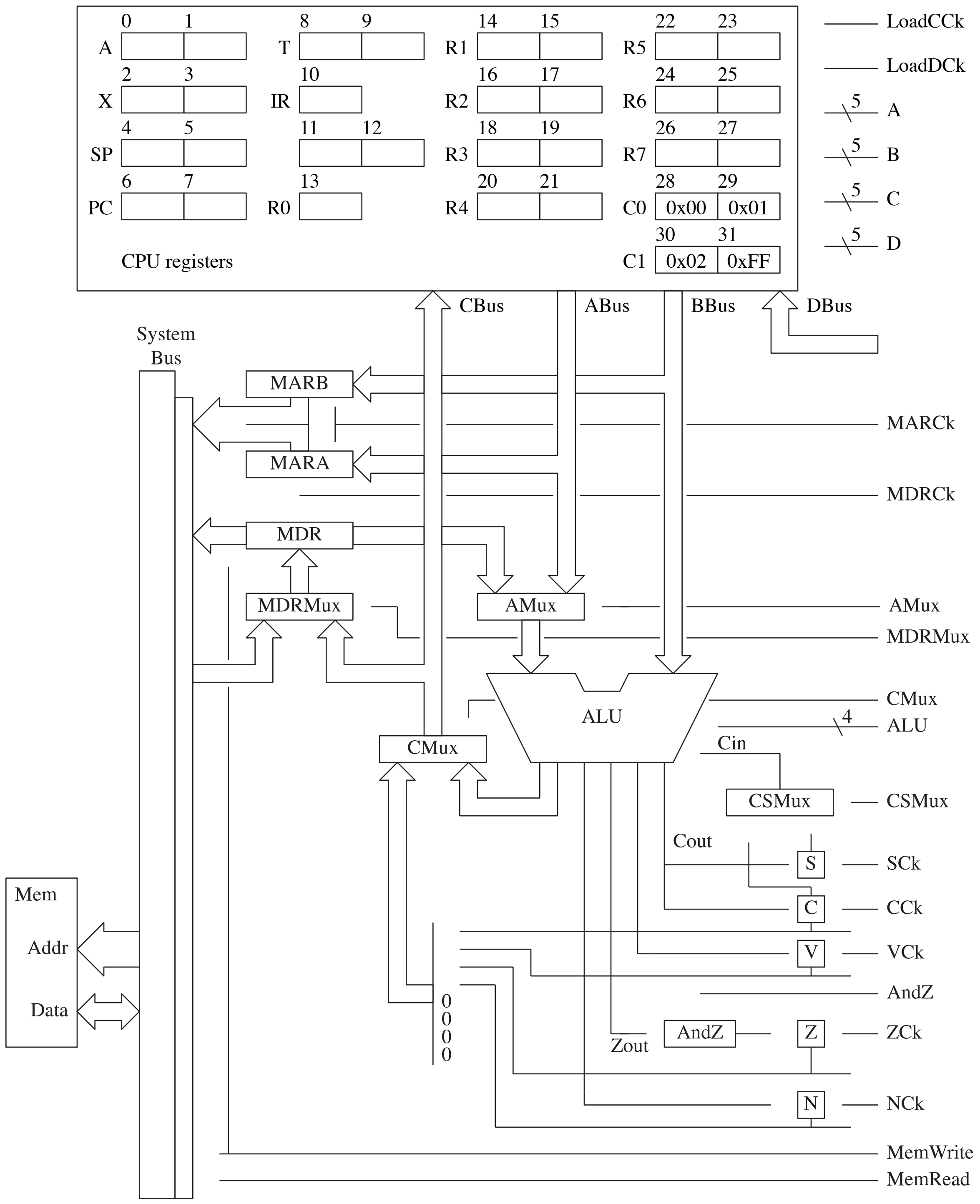
| Instruction Specifier | Mnemonic | Instruction | Addres sing Mode | Status Bits |
|---|---|---|---|---|
| 0000 0000 | `RET` | Return from `CALL` | U | |
| 0000 0001 | `SRET` | Return from system `CALL` | U | |
| 0000 0010 | `MOVSPA` | Move SP to A | U | |
| 0000 0011 | `MOVASP` | Move A to SP | U | |
| 0000 0100 | `MOVFLGA` | Move NZVC flags to A[12 : 15] | U | |
| 0000 0101 | `MOVAFLG` | Move A[12 : 15] to NZVC flags | U | NZVC |
| 0000 0110 | `MOVTA` | Move T to A | U | |
| 0000 0111 | `USCALL` | Unary system call | U | |
| 0000 1000 | `NOP` | No operation | 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 | `SCALL` | System call | i, d, n, s, sf, x, sx, sfx | |
| 0011 1aaa | `LDWT` | Load word T from memory | i | |
| 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` | $\text{PC} \leftarrow \text{Mem[SP]}\ ;\ \text{SP} \leftarrow \text{SP} + 2$ |
| `SRET` | $\text{NZVC} \leftarrow \text{Mem[SP]}[4:7]\ ;\ \text{A} \leftarrow \text{Mem}[\text{SP}+1]\ ;\ \text{X} \leftarrow \text{Mem}[\text{SP}+3]\ ;\ \text{PC} \leftarrow \text{Mem}[\text{SP}+5]\ ;\ \text{SP} \leftarrow \text{Mem}[\text{SP}+7]$ |
| `MOVSPA` | $\text{A} \leftarrow \text{SP}$ |
| `MOVASP` | $\text{SP} \leftarrow \text{A}$ |
| `MOVFLGA` | $\text{A}[8:11] \leftarrow 0\ ,\ \text{A}[12:15] \leftarrow \text{NZVC}$ |
| `MOVAFLG` | $\text{NZVC} \leftarrow \text{A}[12:15]$ |
| `MOVTA` | $\text{A} \leftarrow \text{T}$ |
| `USCALL` | $\text{Y} \leftarrow \text{Mem[FFF0]}\ ;\ \text{Mem}[\text{Y}-2] \leftarrow \text{SP}\ ;\ \text{Mem}[\text{Y}-4] \leftarrow \text{PC}\ ;\ \text{Mem}[\text{Y}-6] \leftarrow \text{X}\ ;$ $\text{Mem}[\text{Y}-8] \leftarrow \text{A}\ ;\ \text{Mem}[\text{Y}-9][4:7] \leftarrow \text{NZVC}\ ;\ \text{SP} \leftarrow \text{Y}-9\ ;\ \text{PC} \leftarrow \text{Mem[FFFE]}$ |
| `NOP` | {No operation} |
| | |
| `NOTr` | $\text{r} \leftarrow \neg\text{r}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0$ |
| `NEGr` | $\text{r} \leftarrow -\text{r}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0\ ,\ \text{V} \leftarrow \{\text{overflow}\}$ |
| `ASLr` | $\text{C} \leftarrow \text{r}[0]\ ,\ \text{r}[0:14] \leftarrow \text{r}[1:15]\ ,\ \text{r}[15] \leftarrow 0\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0\ ,\ \text{V} \leftarrow \{\text{overflow}\}$ |
| `ASRr` | $\text{C} \leftarrow \text{r}[15]\ ,\ \text{r}[1:15] \leftarrow \text{r}[0:14]\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0$ |
| `ROLr` | $\text{C} \leftarrow \text{r}[0]\ ,\ \text{r}[0:14] \leftarrow \text{r}[1:15]\ ,\ r[15] \leftarrow \text{C}$ |
| `RORr` | $\text{C} \leftarrow \text{r}[15]\ ,\ \text{r}[1:15] \leftarrow \text{r}[0:14]\ ,\ r[0] \leftarrow \text{C}$ |
| | |
| `BR` | $\text{PC} \leftarrow \text{Oprnd}$ |
| `BRLE` | $\text{N} = 1 \vee \text{Z} = 1 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `BRLT` | $\text{N} = 1 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `BREQ` | $\text{Z} = 1 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `BRNE` | $\text{Z} = 0 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `BRGE` | $\text{N} = 0 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `BRGT` | $\text{N} = 0 \wedge \text{Z} = 0 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `BRV` | $\text{V} = 1 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `BRC` | $\text{C} = 1 \Rightarrow \text{PC} \leftarrow \text{Oprnd}$ |
| `CALL` | $\text{SP} \leftarrow \text{SP} - 2\ ;\ \text{Mem[SP]} \leftarrow \text{PC}\ ;\ \text{PC} \leftarrow \text{Oprnd}$ |
| | |
| `SCALL` | $\text{Y} \leftarrow \text{Mem[FFF0]}\ ;\ \text{Mem}[\text{Y}-1] \leftarrow \text{IR}[0:7]\ ;\ \text{Mem}[\text{Y}-3] \leftarrow \text{SP}\ ;\ \text{Mem}[\text{Y}-5] \leftarrow \text{PC}\ ;\ \text{Mem}[\text{Y}-7] \leftarrow \text{X}\ ;$ $\text{Mem}[\text{Y}-9] \leftarrow \text{A}\ ;\ \text{Mem}[\text{Y}-10][4:7] \leftarrow \text{NZVC}\ ;\ \text{SP} \leftarrow \text{Y}-10\ ;\ \text{PC} \leftarrow \text{Mem[FFFE]}$ |
| `LDWT` | $\text{T} \leftarrow \text{Oprnd}$ |
| | |
| `LDWr` | $\text{r} \leftarrow \text{Oprnd}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0$ |
| `LDBr` | $\text{r}[8:15] \leftarrow \text{byte Oprnd}\ ;\ \text{N} \leftarrow 0\ ,\ \text{Z} \leftarrow \text{r}[8:15] = 0$ |
| `STWr` | $\text{Oprnd} \leftarrow \text{r}$ |
| `STBr` | $\text{byte Oprnd} \leftarrow \text{r}[8:15]$ |
| | |
| `CPWr` | $\text{Y} \leftarrow \text{r} - \text{Oprnd}\ ;\ \text{N} \leftarrow \text{Y} < 0\ ,\ \text{Z} \leftarrow \text{Y} = 0\ ,\ \text{V} \leftarrow \{\text{overflow}\}\ ,\ \text{C} \leftarrow \{\text{carry}\}\ ;\ \text{N} \leftarrow \text{N} \oplus \text{V}$ |
| `CPBr` | $\text{Y} \leftarrow \text{r}[8:15] - \text{byte Oprnd}\ ;\ \text{N} \leftarrow \text{Y} < 0\ ,\ \text{Z} \leftarrow \text{Y} = 0\ ,\ \text{V} \leftarrow 0\ ,\ \text{C} \leftarrow 0$ |
| | |
| `ADDr` | $\text{r} \leftarrow \text{r} + \text{Oprnd}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0\ ,\ \text{V} \leftarrow \{\text{overflow}\}\ ,\ \text{C} \leftarrow \{\text{carry}\}$ |
| `SUBr` | $\text{r} \leftarrow \text{r} - \text{Oprnd}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0\ ,\ \text{V} \leftarrow \{\text{overflow}\}\ ,\ \text{C} \leftarrow \{\text{carry}\}$ |
| `ANDr` | $\text{r} \leftarrow \text{r} \wedge \text{Oprnd}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0$ |
| `ORr` | $\text{r} \leftarrow \text{r} \vee \text{Oprnd}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0$ |
| `XORr` | $\text{r} \leftarrow \text{r} \oplus \text{Oprnd}\ ;\ \text{N} \leftarrow \text{r} < 0\ ,\ \text{Z} \leftarrow \text{r} = 0$ |
| | |
| `ADDSP` | $\text{SP} \leftarrow \text{SP} + \text{Oprnd}$ |
| `SUBSP` | $\text{SP} \leftarrow \text{SP} - \text{Oprnd}$ |

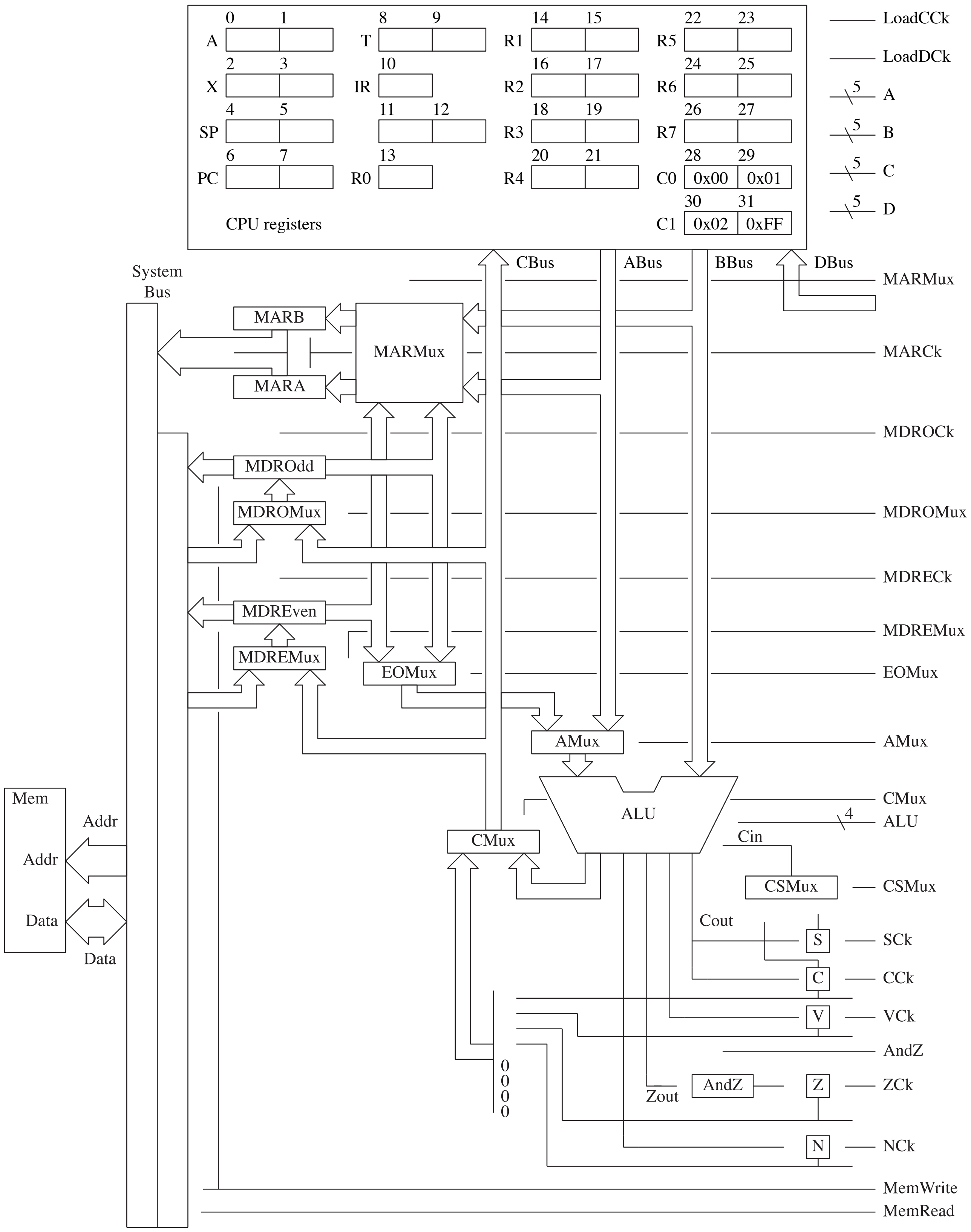
Here is the memory map of the Pep/10 system. The shaded portion is ROM. Compared to the Pep/9 memory map, this map has several additional components – a disk input port at address 3333, a power off port at address 6666, and a dispatcher at address 7777. Pep/9 has six machine vectors from FFF4 to FFFE. Because of the new components, Pep/10 has nnine machine vectors at addresses FFEE 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.


CPU registers
A 0 1
X 2 3
SP 4 5
PC 6 7
T 8 9
IR 10
11 12
R0 13
R1 14 15
R2 16 17
R3 18 19
R4 20 21
R5 22 23
R6 24 25
R7 26 27
C0 28 29 0x00 0x01
C1 30 31 0x02 0xFF
LoadCCk
LoadDCk
5 A
5 B
5 C
5 D
CBus
ABus
BBus
DBus
System Bus
MARB
MARA
MARMux
MDROdd
MDROMux
MDREven
MDREMux
EOMux
AMux
ALU
CMux
Cin
CSMux
Cout
S
C
V
Z
N
AndZ
Zout
0
0
0
0
Mem
Addr
Data
MARMux
MARCk
MDROCk
MDROMux
MDRECk
MDREMux
EOMux
AMux
CMux
4 ALU
CSMux
SCk
CCk
VCk
AndZ
ZCk
NCk
MemWrite
MemRead