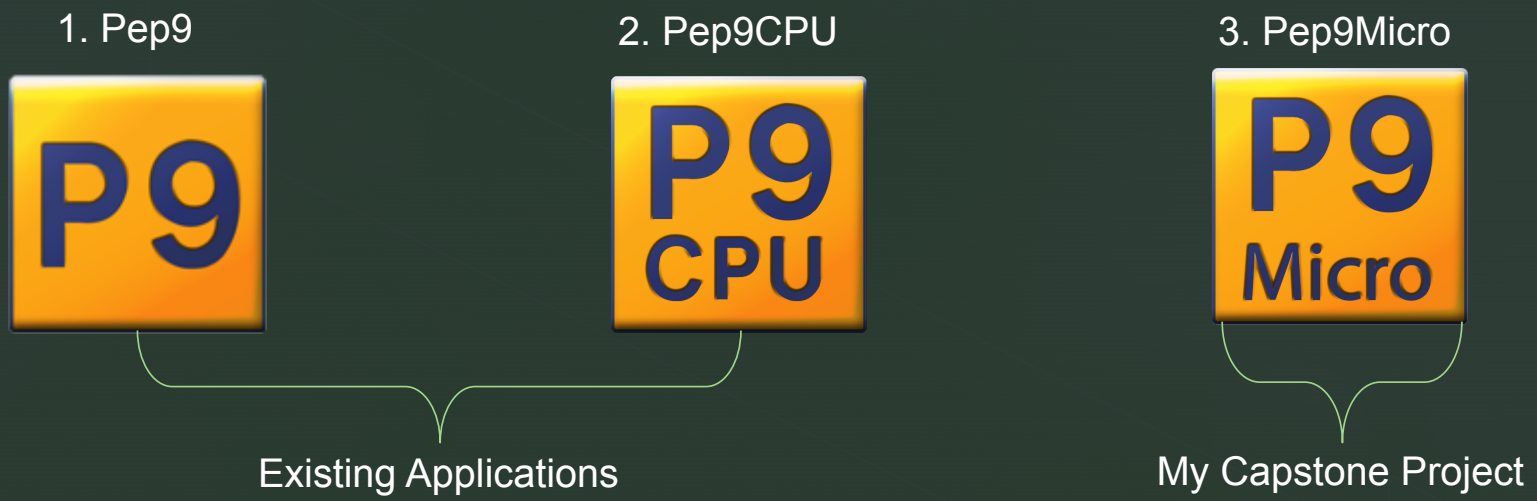
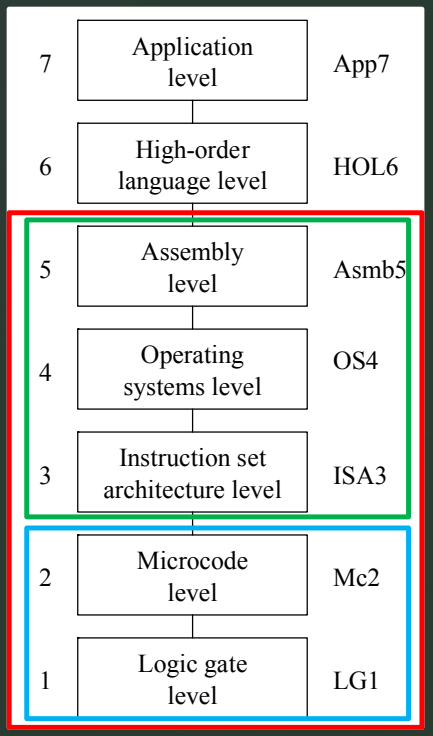
Matthew McRaven

Pep9Micro:
Merging abstractions

Roadmap to Pep9Micro

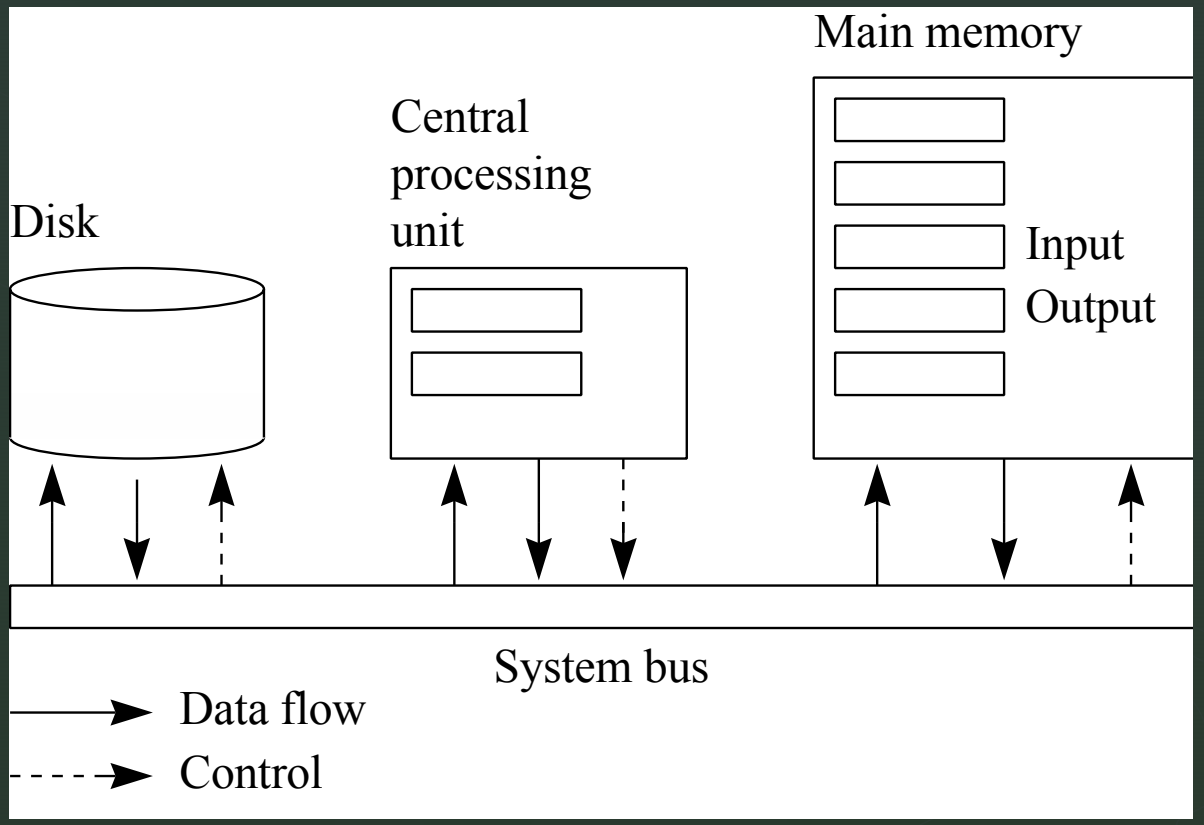


Layers of Abstraction



- Layers spanned by:
  - Pep9
  - Pep9CPU
  - Pep9Micro
- Pep9 & Pep9CPU do not overlap.
- Pep9Micro joins the functionality of both.

Pep9's hardware abstraction



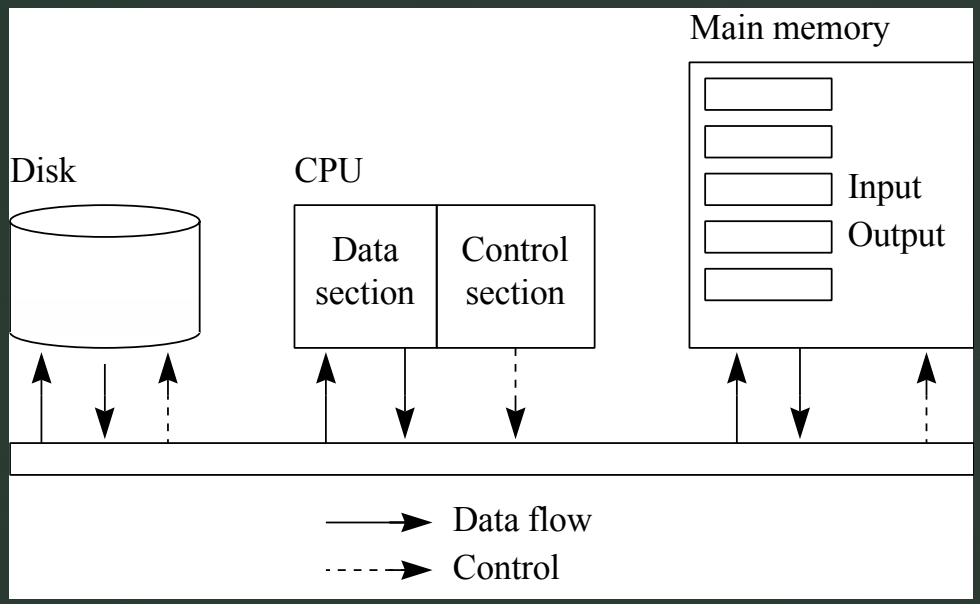
A Pep9 Assembly Program Demo

- Simple looping program to print “0 1 2 Done!”
- Examine program in Pep9.

```
; Program that prints integers in [0, 2] via a loop.
        LDBX    ' ', i
        LDWA    iter, d
; Start of loop, print out iterator.
loop:   DECO    iter, d
; Print a space between numbers.
        STBX    charOut, d
; Increment iterator.
        ADDA    1, i
        STWA    iter, d
; Execute loop again if iterator <  3.
        CPWA    3, i
        BRLT    loop
; Print out "Done!", terminate program.
        STRO    msg, d
        STOP
iter:   .WORD   0
msg:    .ASCII  "\nDone!\x00"
        .END
```

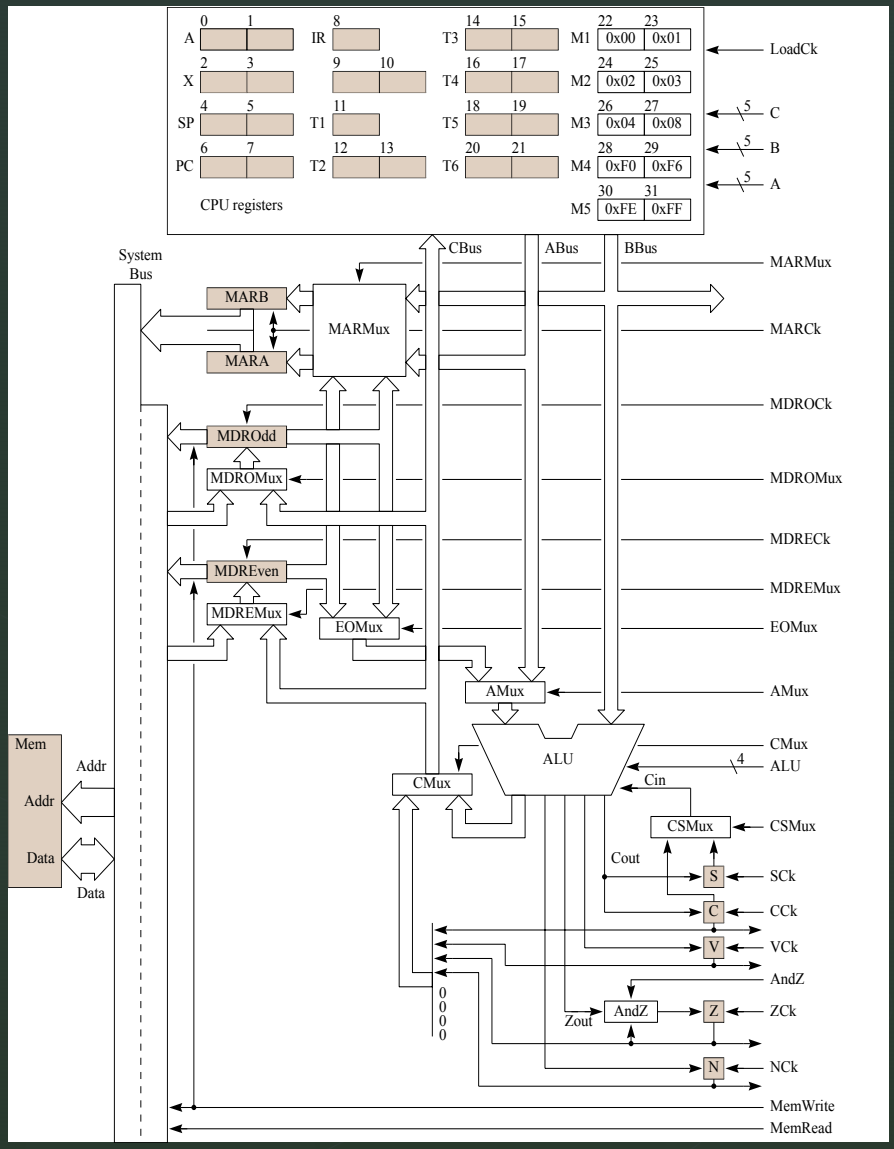
Pep9CPU's hardware abstraction

- Pep9's CPU is split into two parts.
- Data section executes.
- Control section decides.



CPU Data Section

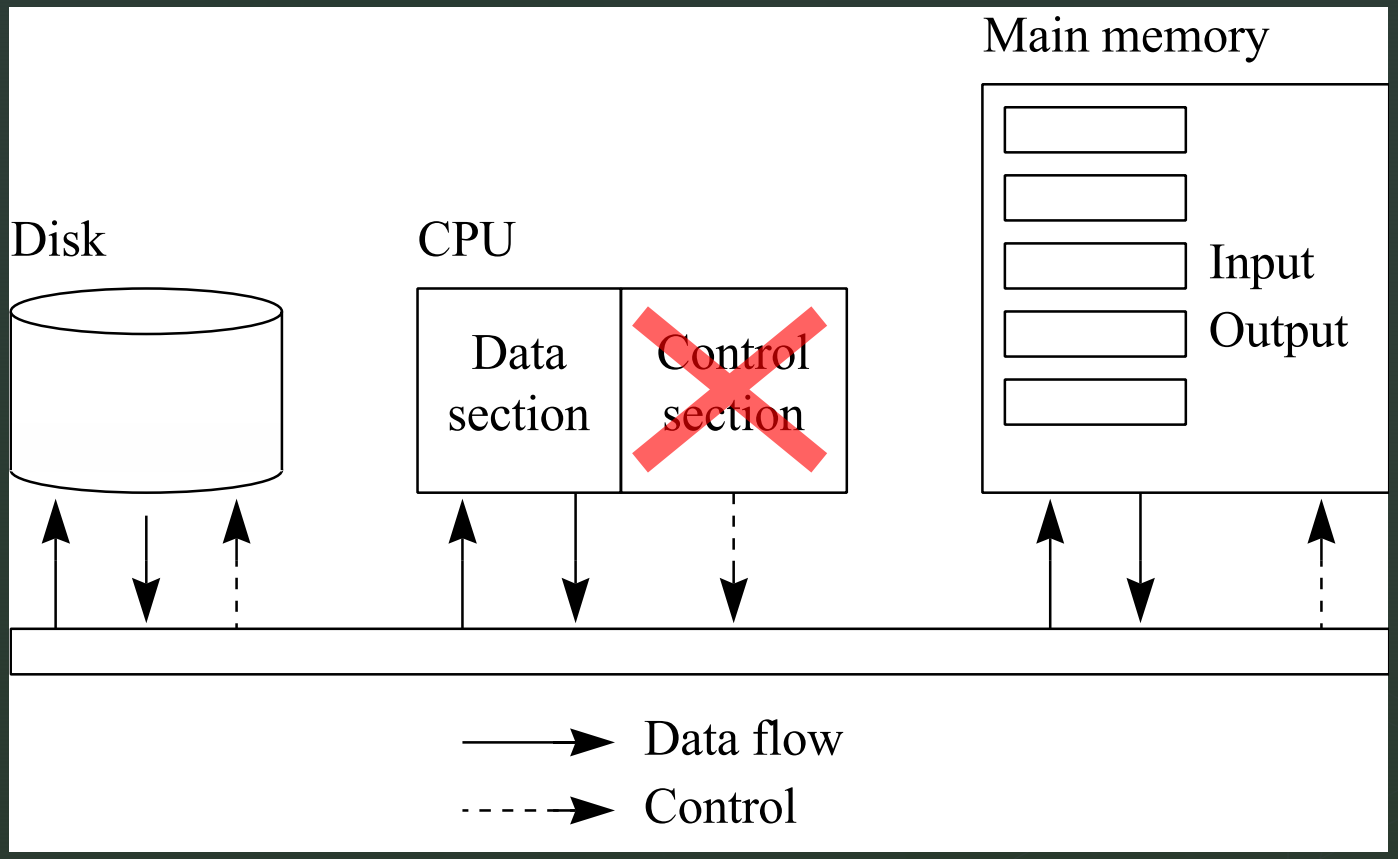
- ”Zoomed in” on data section of CPU
- Each box represent a circuit.
- Lines & arrows show data flow.



A Pep9CPU sample microprogram

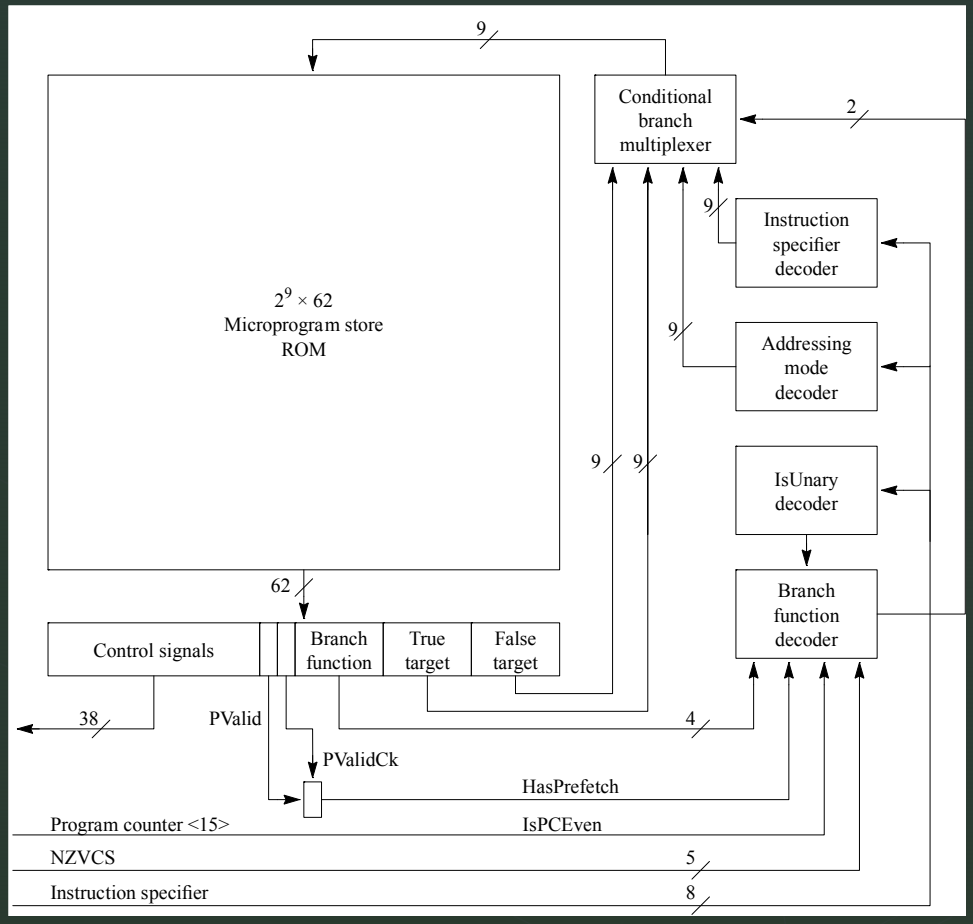
```
// Fetch the instruction specifier and increment PC by 1
// Fetch & decode the operand specified, increment PC by 2.
// ADDA 1 ,i
// RTL: A <- A + Oprnd; N <- A<0, Z <- A=0, V <- {overflow}, C <- {carry}
// Immediate addressing: Oprnd = OprndSpec

UnitPre: IR=0x000000, PC=0x000C, Mem[0x000C]=0x6000,Mem[0x000e]=0x01, S=0, A=0x000
UnitPost: IR=0x600001, PC=0x000f, A=0x0003

// MAR <- PC.
A=6, B=7, marmux=1; MARCk
// Fetch instruction specifier, first byte of operand, PC <- PC + 2.
MemRead, A=7, B=24, AMux=1, ALU=1, CMux=1, C=7; SCk, LoadCk
MemRead, A=6, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=6; LoadCk
MemRead, mdremux=0, mdromux=0; mdrock, mdreck
// IR <- instruction specifier.
AMux=0, eomux=0, ALU=0, CMux=1, C=8; LoadCk

// MAR <- PC. Oprnd<hi> <- Mem[PC - 1]
A=6, B=7, marmux=1, eomux=1, amux=0, alu=0, cmux=1, c=9; MARCk,loadck
// Fetch second byte of operand, PC <- PC + 1.
MemRead, A=7, B=23, AMux=1, ALU=1, CMux=1, C=7; SCk, LoadCk
MemRead, A=6, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=6; LoadCk
MemRead, mdremux=0; mdreck

// MAR <- PC. Oprnd<lo> <- Mem[PC - 1]
eomux=0, amux=0, alu=0, cmux=1, c=10; loadck

// A<low> <- A<low> + Oprnd<low>, Save shadow carry.
A=1, B=10, AMux=1, ALU=1, AndZ=0, CMux=1, C=1; ZCk, SCk, LoadCk
// A<high> <- A<high> plus Oprnd<high> plus saved carry.
A=0, B=9, AMux=1, CSMux=1, ALU=2, AndZ=1, CMux=1, C=0; NCk, ZCk, VCk, CCk, LoadCk
```

Pep9CPU is *NOT* a computer



Creating Pep9Micro

- First, must design control circuitry.
- Complete control section allows arbitrary program execution.

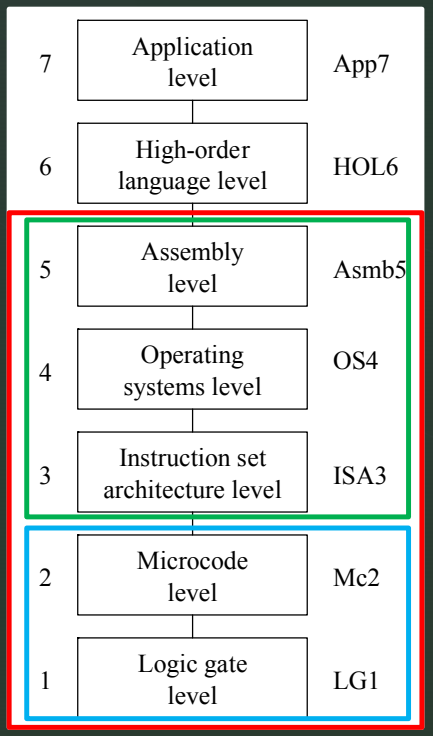


Using Pep9Micro to simulate multiple levels of abstraction

```
; Program that prints integers in [0, 2] via a loop.
          LDBX    ' ', i
          LDWA    iter, d
; Start of loop, print out iterator.
loop:     DECO    iter, d
; Print a space between numbers.
          STBX    charOut, d
; Increment iterator.
          ADDA    1, i
          STWA    iter, d
; Execute loop again if iterator <  3.
          CPWA    3, i
          BRLT    loop
; Print out "Done!", terminate program.
          STRO    msg, d
          STOP
iter:     .WORD   0
msg:      .ASCII  "\nDone!\x00"
          .END
```

- Same program as before.
- Examine software & hardware in Pep9Micro.

Layers of Abstraction



- Layers spanned by:
  - Pep9
  - Pep9CPU
  - Pep9Micro
- Pep9Micro allows program visualization in hardware.

Matthew McRaven

Pep9Micro:
Merging abstractions

https://github.com/StanWarford/pep9suite