Pep9Milli

Symbolic Verification of a CISC Processor

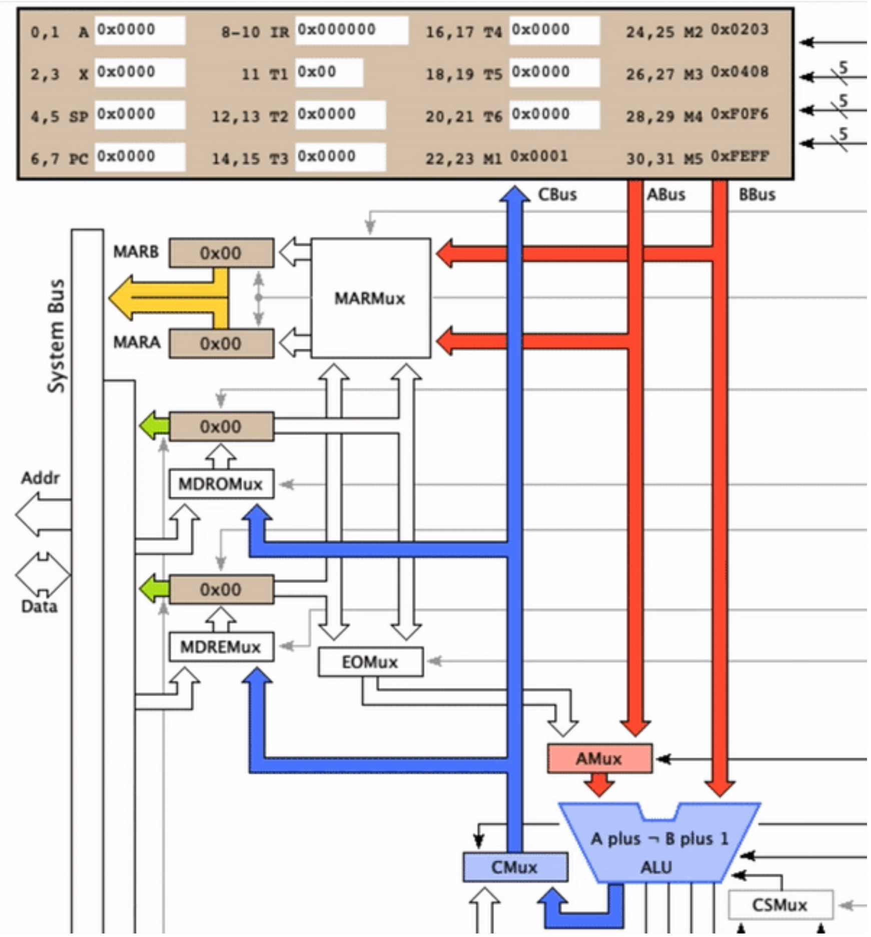
Matthew McRaven
05 December 2019

Overview

- Describe Pep/9, the processor being verified
- Motivate and describe new hardware language: *millicode*
- Discuss verification framework and results

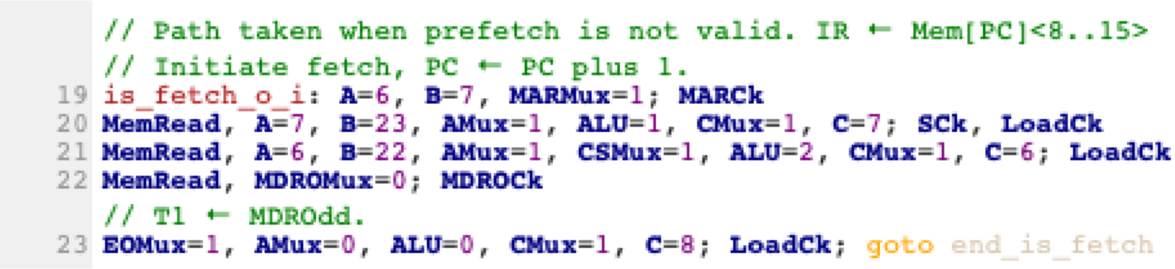
Pep/9 Overview

- Pedagogical virtual computer
- 16-bit CISC computer
- Simulated at various levels of abstraction
  - ○ Assembly Language
  - ○ Operating System
  - ○ Hardware Control, termed *microcode*



Improvements in Pep9Micro

- Feature disparity between assembler and microcode
- Designed CPU control section, completing processor
- Correct in all circumstances?

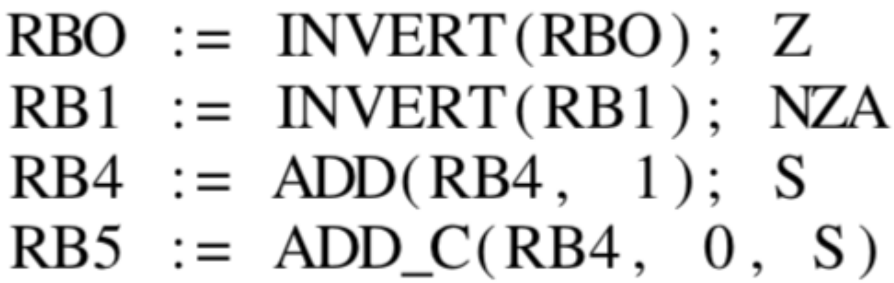


```
// Path taken when prefetch is not valid. IR ← Mem[PC]<8..15>
// Initiate fetch, PC ← PC plus 1.
is_fetch_o_i: A=6, B=7, MARMux=1; MARCk
MemRead, A=7, B=23, AMux=1, ALU=1, CMux=1, C=7; SCk, LoadCk
MemRead, A=6, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=6; LoadCk
MemRead, MDROMux=0; MDROCk
// T1 ← MDROdd.
EOMux=1, AMux=0, ALU=0, CMux=1, C=8; LoadCk; goto end_is_fetch
```

Industry Verification Experience

Vendor	**Technique**
Centaur	Formal
IBM	Functional
Intel	Formal
Rockwell	Symbolic

A Different Direction

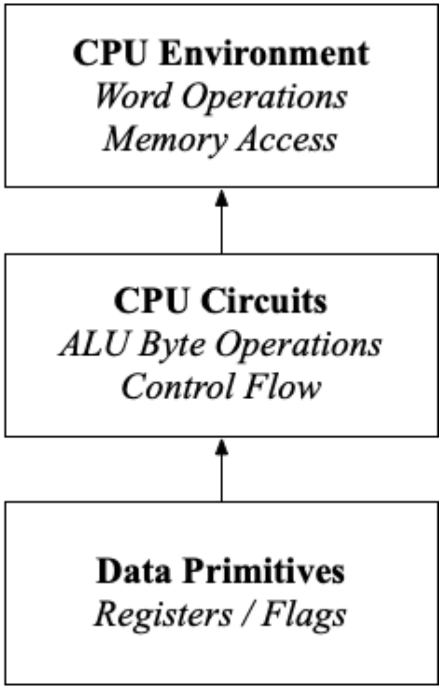


```
RBO  :=  INVERT(RBO);  Z
RB1  :=  INVERT(RB1);  NZA
RB4  :=  ADD(RB4,  1);  S
RB5  :=  ADD_C(RB4,  0,  S)
```

- No VHDL/Verilog description
- Microcode is hard to read
- Enter *millicode*, a new hardware control language
- Translates to microcode, verifiable C

Constructing a Verification Environment

- Verification needs model of CPU
- Layered model for abstraction
- Models CPU operations and memory
- Translate millicode to C interface



CPU Environment
Word Operations
Memory Access

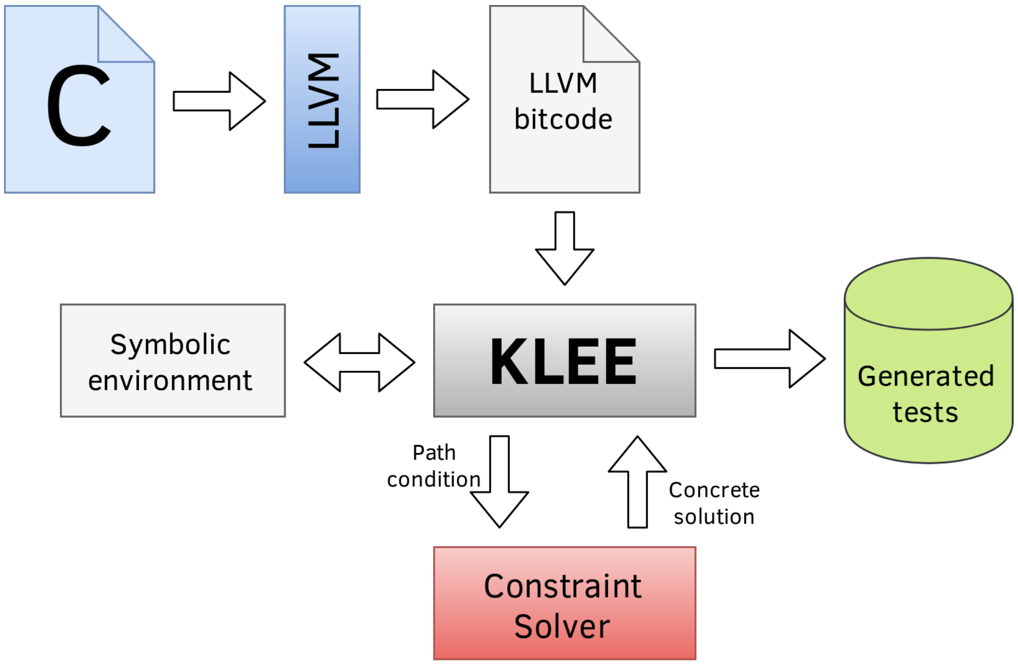
CPU Circuits
ALU Byte Operations
Control Flow

Data Primitives
Registers / Flags

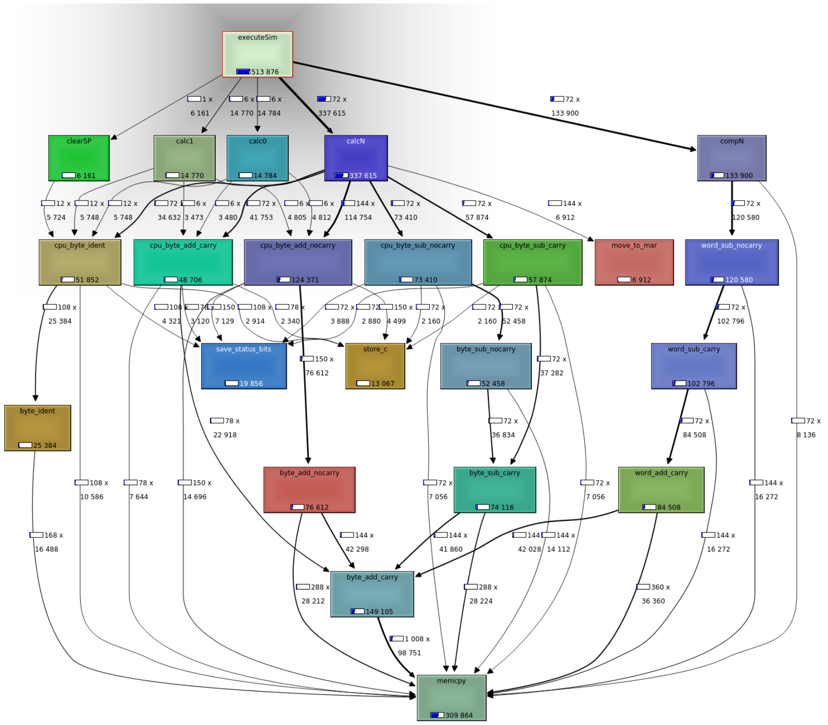
```
RB0 := ADD(RB5, RB6) ≡ cpu_add(cpu, 5, 6, 0)
```

Applying Verification

- Klee performs symbolic execution on C
- Manually insert assertions
- Run Klee, check for assertion errors



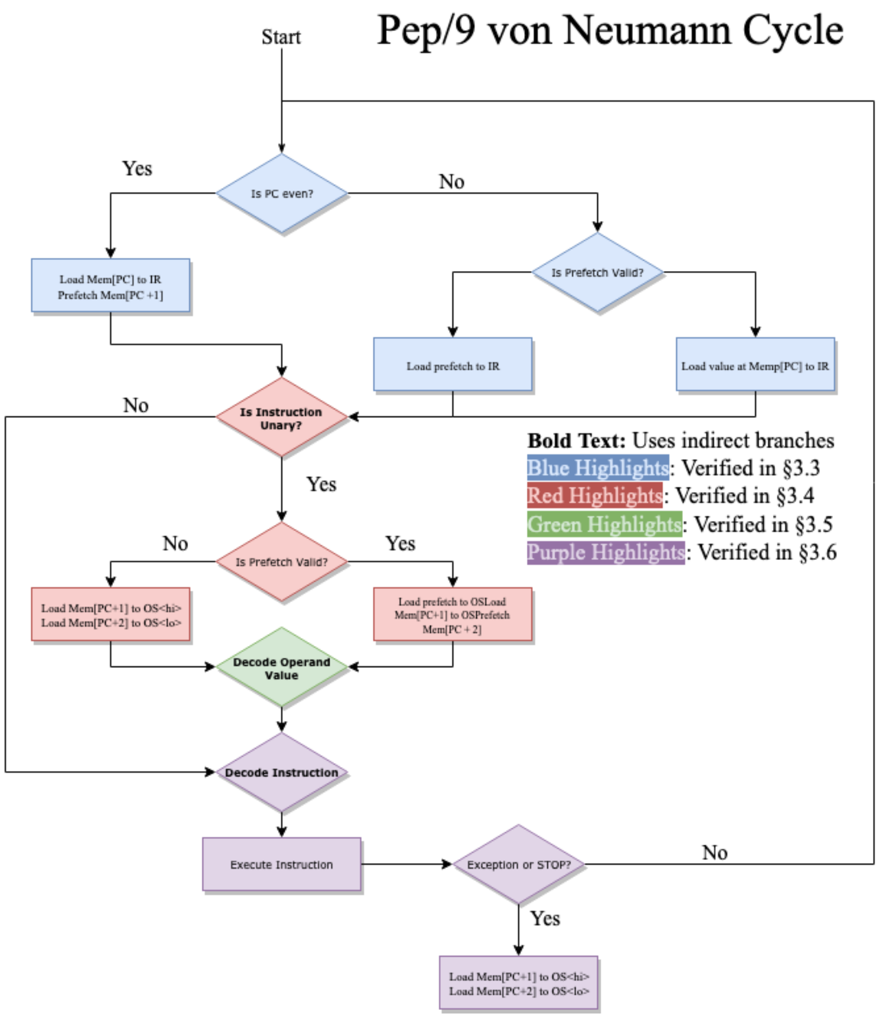
Verifying a Trivial Program



- Compute first 14 Fibonacci numbers
- Test millicode, verification environment
- Manual verification conditions
- Verified successfully in 63 seconds
  - Verification call tree (see left)

What Needs Verifying?

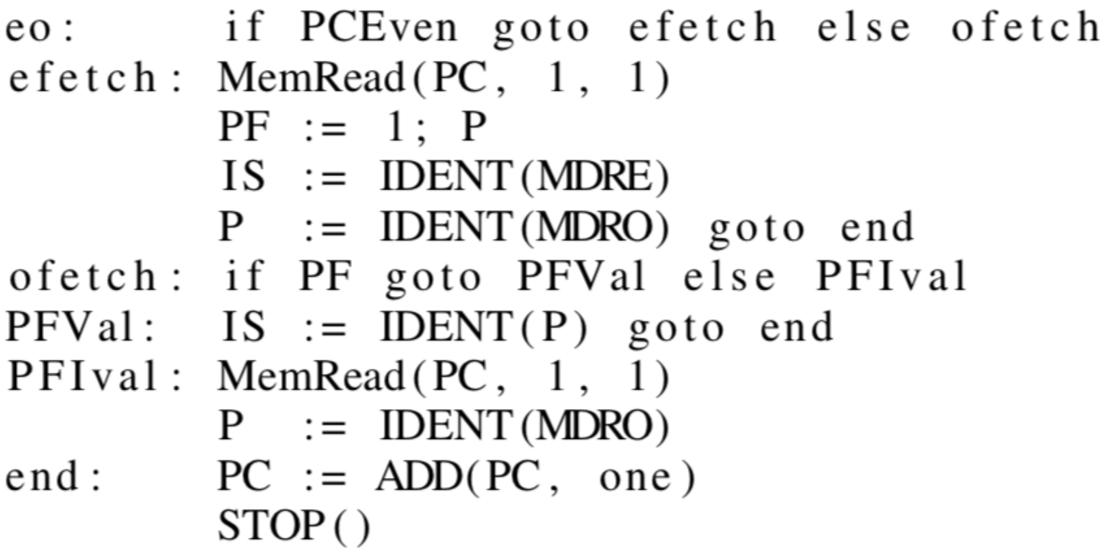
- Verify hardware implements instruction set
- Hardware broken into 4 units:
  - ○ Instruction Fetch
  - ○ Operand Fetch
  - ○ Operand Decode
  - ○ Instruction Execute
- Analyze all 4 units to verify Pep/9 processor


Pep/9 von Neumann Cycle

Bold Text: Uses indirect branches
Blue Highlights: Verified in §3.3
Red Highlights: Verified in §3.4
Green Highlights: Verified in §3.5
Purple Highlights: Verified in §3.6

Instruction Fetch

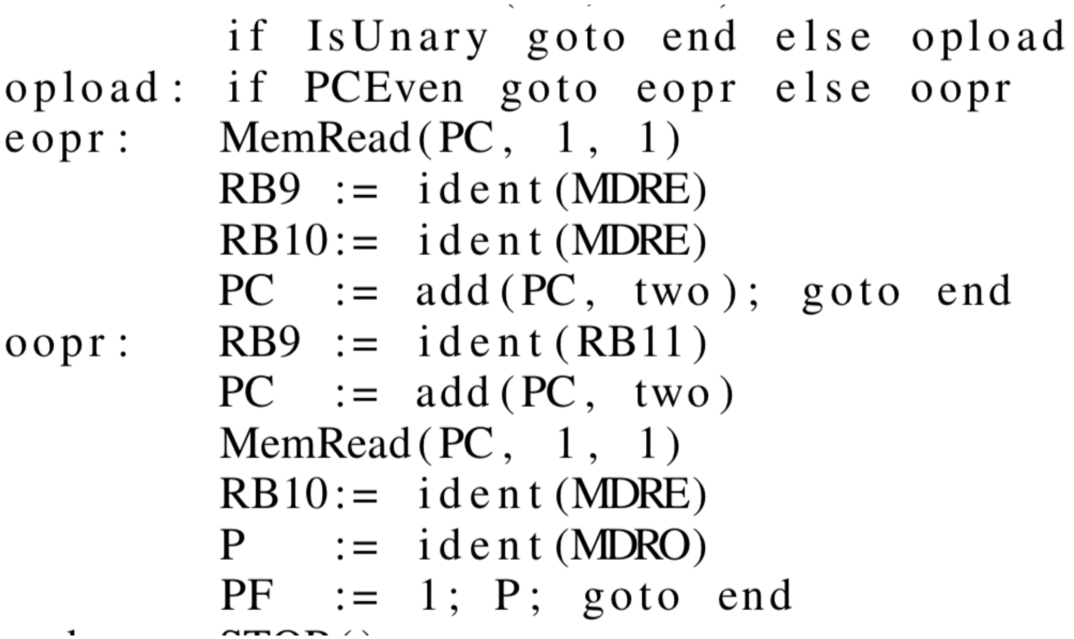
- Loads instructions
- Verification: success!
- 10 unique paths in 1 second



```
eo:      if PCEven goto efetch else ofetch
efetch:  MemRead(PC, 1, 1)
         PF := 1; P
         IS := IDENT(MDRE)
         P  := IDENT(MDRO) goto end
ofetch:  if PF goto PFVal else PFIval
PFVal:   IS := IDENT(P) goto end
PFIval:  MemRead(PC, 1, 1)
         P  := IDENT(MDRO)
end:     PC := ADD(PC, one)
         STOP()
```

Operand Fetch

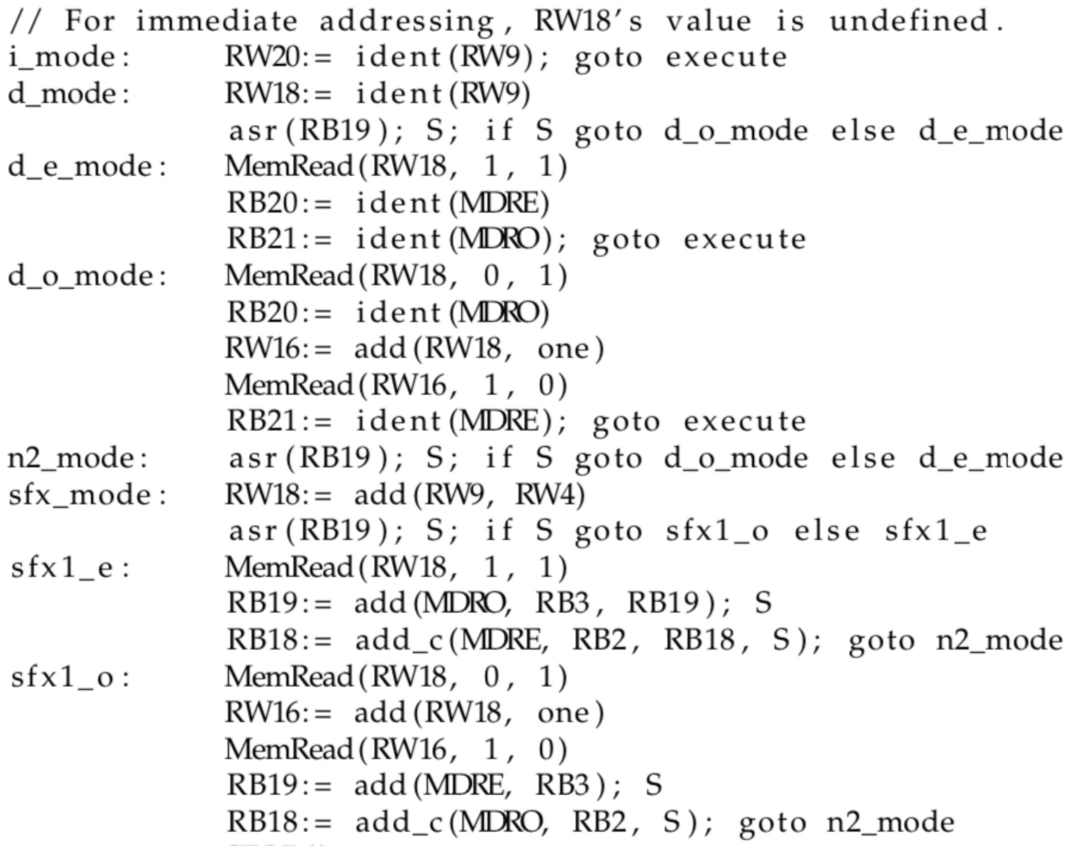
- Loads non-unary instruction operands
- Verification: success!
- 16 unique paths in 3 seconds



```
         if IsUnary goto end else opload
opload:  if PCEven goto eopr else oopr
eopr:    MemRead(PC, 1, 1)
         RB9  := ident(MDRE)
         RB10:= ident(MDRE)
         PC   := add(PC, two); goto end
oopr:    RB9  := ident(RB11)
         PC   := add(PC, two)
         MemRead(PC, 1, 1)
         RB10:= ident(MDRE)
         P    := ident(MDRO)
         PF   := 1; P; goto end
```

Operand Decode

- Converts operand, addressing mode to useful value
- Verification: success!
- 4,026 unique paths in 4 hours
- **Victim of state space explosion**



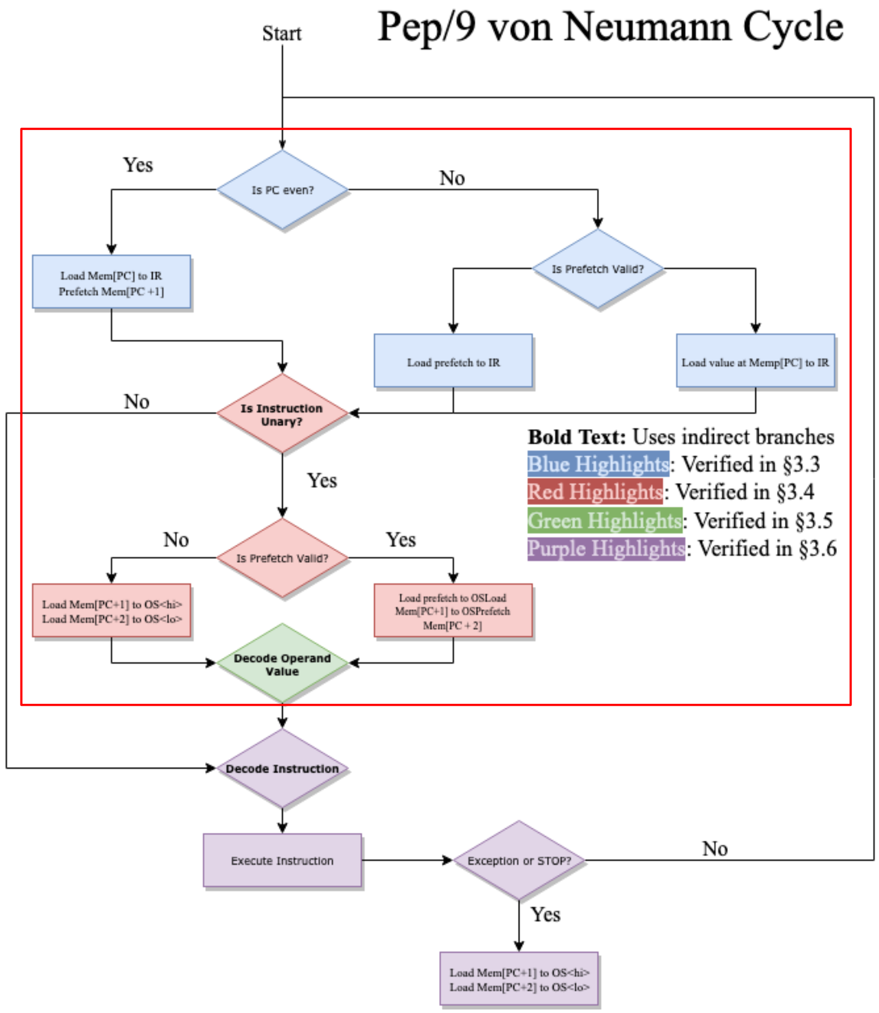
```
// For immediate addressing, RW18's value is undefined.
i_mode:     RW20:= ident(RW9); goto execute
d_mode:     RW18:= ident(RW9)
            asr(RB19); S; if S goto d_o_mode else d_e_mode
d_e_mode:   MemRead(RW18, 1, 1)
            RB20:= ident(MDRE)
            RB21:= ident(MDRO); goto execute
d_o_mode:   MemRead(RW18, 0, 1)
            RB20:= ident(MDRO)
            RW16:= add(RW18, one)
            MemRead(RW16, 1, 0)
            RB21:= ident(MDRE); goto execute
n2_mode:    asr(RB19); S; if S goto d_o_mode else d_e_mode
sfx_mode:   RW18:= add(RW9, RW4)
            asr(RB19); S; if S goto sfx1_o else sfx1_e
sfx1_e:     MemRead(RW18, 1, 1)
            RB19:= add(MDRO, RB3, RB19); S
            RB18:= add_c(MDRE, RB2, RB18, S); goto n2_mode
sfx1_o:     MemRead(RW18, 0, 1)
            RW16:= add(RW18, one)
            MemRead(RW16, 1, 0)
            RB19:= add(MDRE, RB3); S
            RB18:= add_c(MDRO, RB2, S); goto n2_mode
```

Further Research

- Automate millicode translation
- Stricter memory model
- Pep/10 improvements

Conclusion

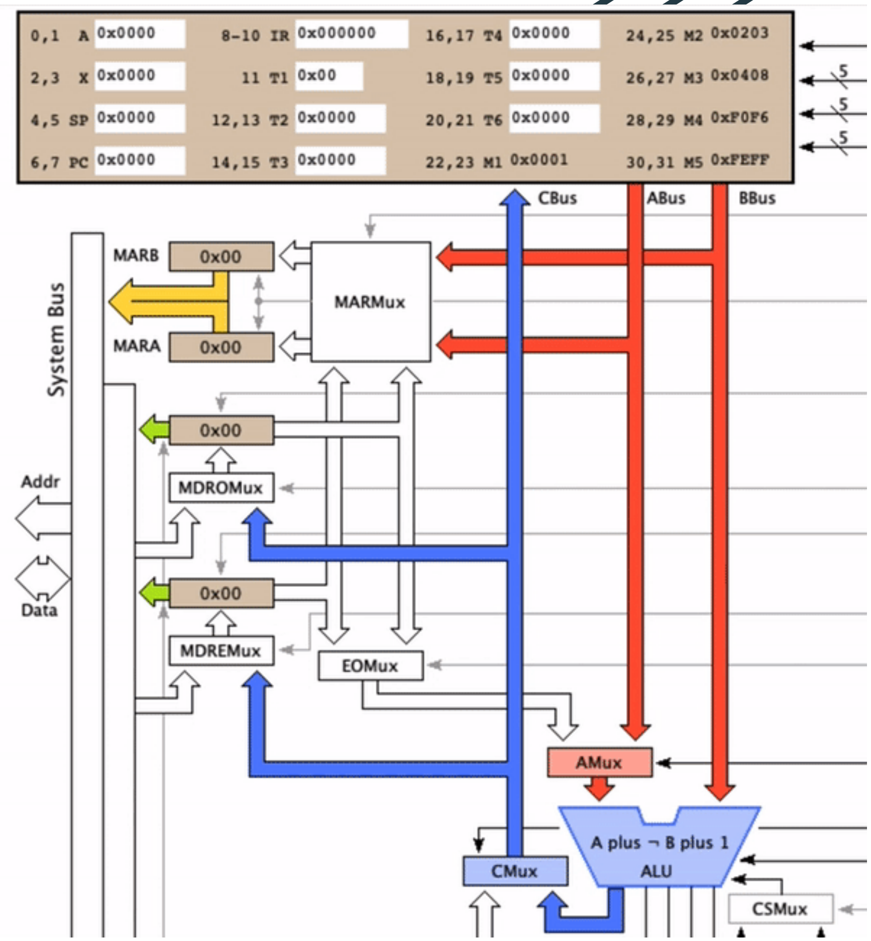
- Introduced hardware control language, *millicode*
- Discussed verification architecture
- Shared multiple verification results
- Verified 3 CPU segments (red box)



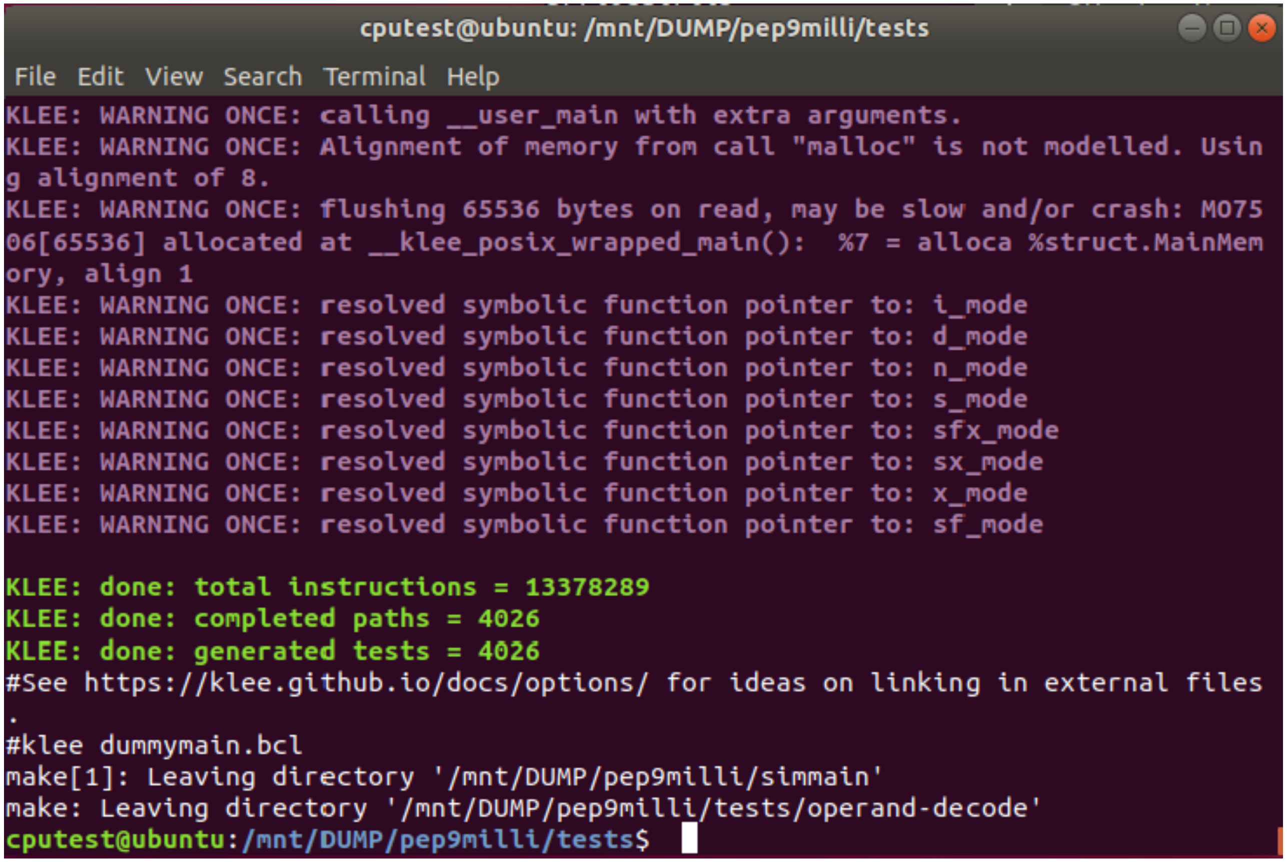
Pep9Milli

Symbolic Verification of a CISC Processor

Matthew McRaven
https://github.com/Matthew-McRaven/pep9milli



0,1	A	0x0000	8-10	IR	0x000000	16,17	T4	0x0000	24,25	M2	0x0203
2,3	X	0x0000	11	T1	0x00	18,19	T5	0x0000	26,27	M3	0x0408
4,5	SP	0x0000	12,13	T2	0x0000	20,21	T6	0x0000	28,29	M4	0xF0F6
6,7	PC	0x0000	14,15	T3	0x0000	22,23	M1	0x0001	30,31	M5	0xFEFF


cputest@ubuntu: /mnt/DUMP/pep9milli/tests
File Edit View Search Terminal Help

```
KLEE: WARNING ONCE: calling __user_main with extra arguments.
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Usin
g alignment of 8.
KLEE: WARNING ONCE: flushing 65536 bytes on read, may be slow and/or crash: MO75
06[65536] allocated at __klee_posix_wrapped_main():  %7 = alloca %struct.MainMem
ory, align 1
KLEE: WARNING ONCE: resolved symbolic function pointer to: i_mode
KLEE: WARNING ONCE: resolved symbolic function pointer to: d_mode
KLEE: WARNING ONCE: resolved symbolic function pointer to: n_mode
KLEE: WARNING ONCE: resolved symbolic function pointer to: s_mode
KLEE: WARNING ONCE: resolved symbolic function pointer to: sfx_mode
KLEE: WARNING ONCE: resolved symbolic function pointer to: sx_mode
KLEE: WARNING ONCE: resolved symbolic function pointer to: x_mode
KLEE: WARNING ONCE: resolved symbolic function pointer to: sf_mode

KLEE: done: total instructions = 13378289
KLEE: done: completed paths = 4026
KLEE: done: generated tests = 4026
#See https://klee.github.io/docs/options/ for ideas on linking in external files
.
#klee dummymain.bcl
make[1]: Leaving directory '/mnt/DUMP/pep9milli/simmain'
make: Leaving directory '/mnt/DUMP/pep9milli/tests/operand-decode'
cputest@ubuntu:/mnt/DUMP/pep9milli/tests$
```

Successful Verification Run by Klee