KAMIL M ROCKI

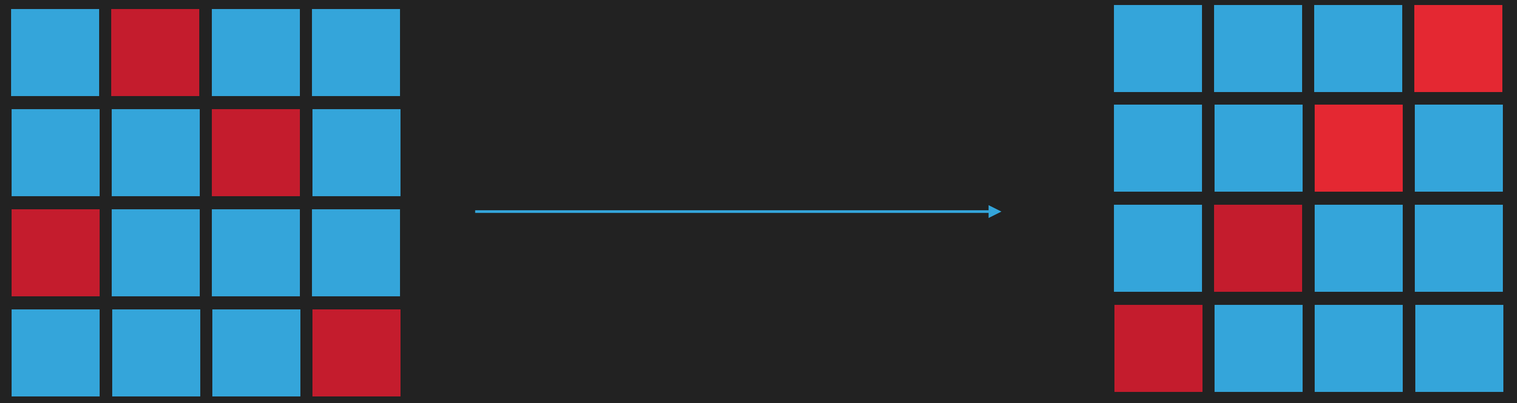
EVOLVING MACHINE CODE

http://olab.is.s.u-tokyo.ac.jp/~kamil.rocki/dca/

GOAL: LEARN PROGRAM GIVEN TARGET MEMORY CONTENT

PROGRAM

SHORT: ~16-32 BYTES



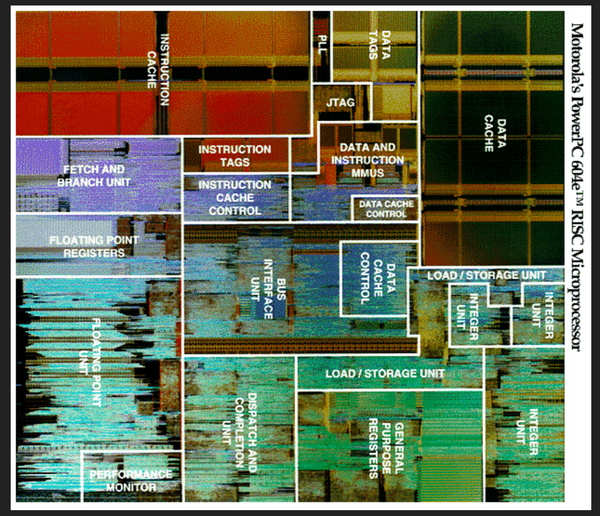
MEMORY

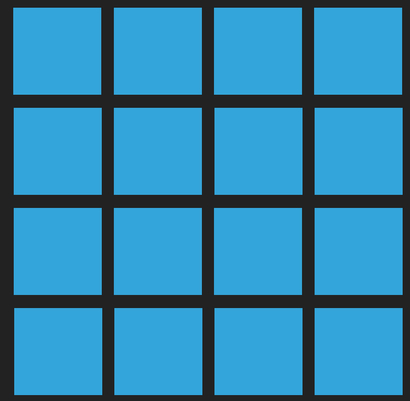
MEMORY

SIMPLIFIED CPU MODEL

PROGRAM

INTERNAL STATE / REGISTERS





VISIBLE/EXTERNAL MEMORY

CUSTOM INSTRUCTION SET

0.013% of XCVU9P (AWS F1)
320 FPGA logic blocks ~ 1/5 of RISC-V32-IE

1: LOAD DEST, SRC, OFFSET

2: STORE DEST, OFFSET, SRC

3: SET, DEST, VAL

4: BGE ARG0, ARG1, OFFSET

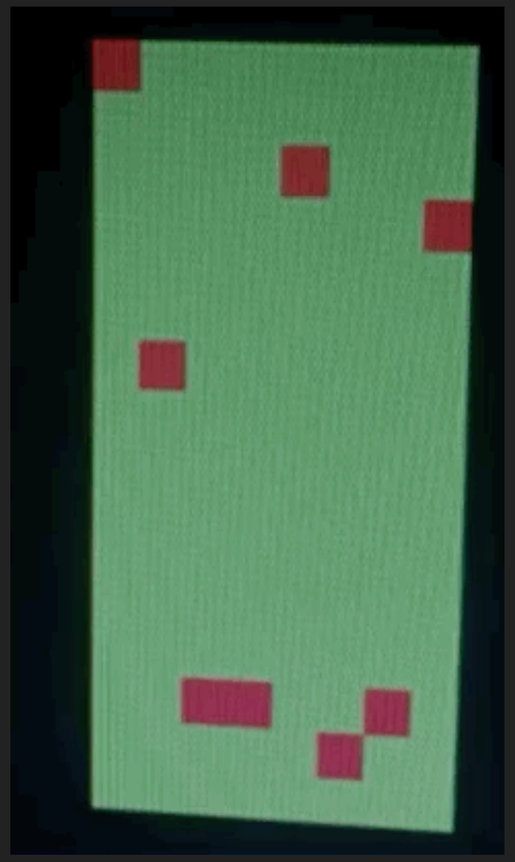
5: ADD DEST, SRC0, SRC1

6: SUB DEST, SRC, VAL

7: HALT

8 x 8-bit registers

fetch, decode and execute in a single cycle



LEARNING DONE IN FPGA

LEARNED SORT

COMPARE

AND SWAP

```
3600 SET %6, 0
3707 SET %7, 0+7
3400 SET %4, 0
3600 SET %6, 0
6571 SUB %5, %7, 1
4561 BGE %5,%6,1 (if  %5>%6 skip next)
7000 HALT
3101 SET %1, 0+1
4542 BGE %5,%4,2 (if  %5>%4 skip next 2)
6771 %7 = %7 - 1
3001  SET %0, 1
1240 LOAD %2<= MEM[%4+0]
1341 LOAD %3<= MEM[%4+1]
4322 BGE %4,%3,1 (if  %3>%2 skip next 2)
2403 STORE MEM[%4+0]<= %3
2412 STORE MEM[%4+1]<= %2
5441 %4 = %4+1
3007 GOTO 07
```

RISC-V

~350 cycles

CUSTOM ISA

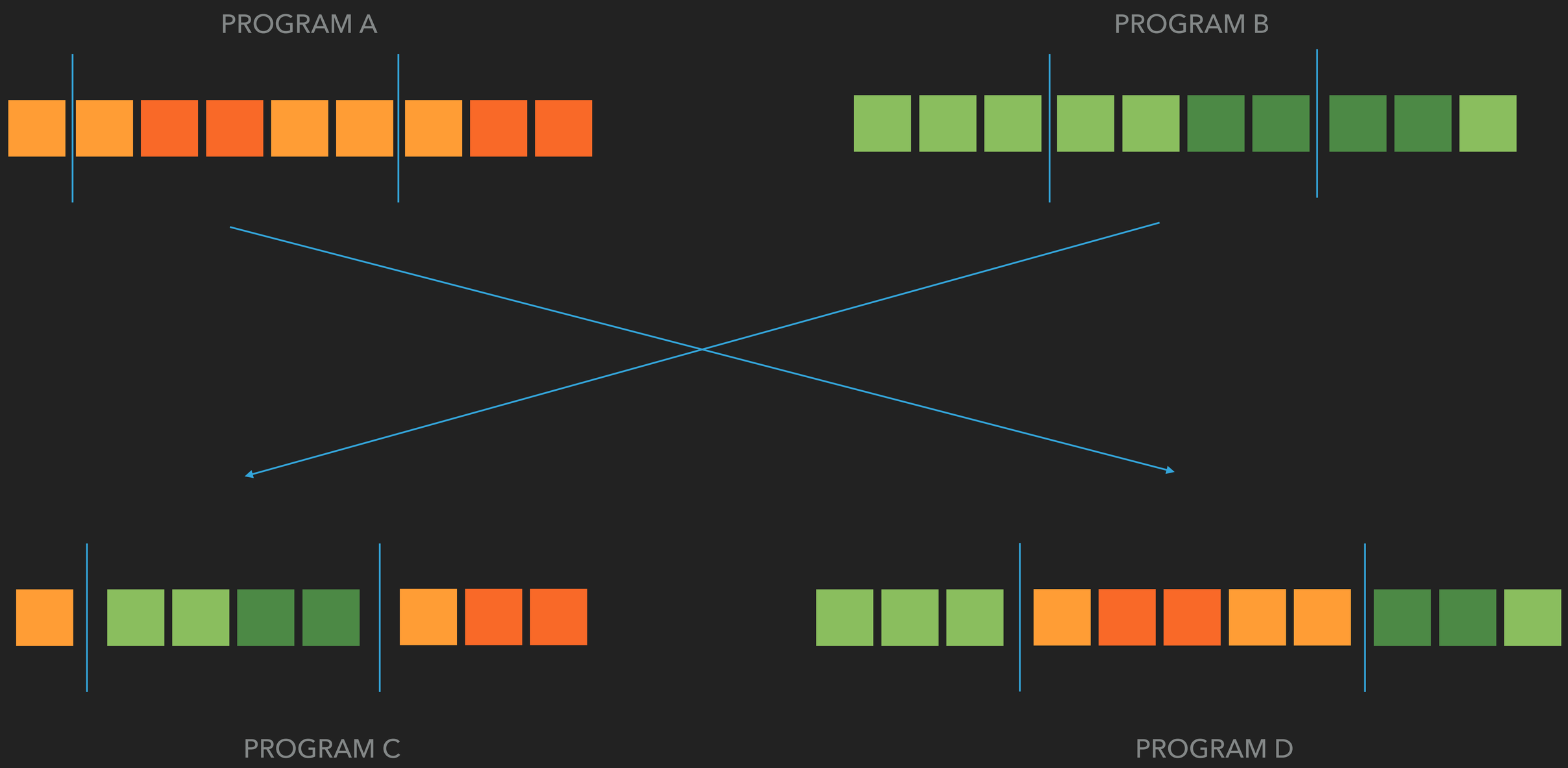
~90 cycles

x86 asm

~900 cycles

x86 python

~80000 cycles


PROGRAM A
PROGRAM B
PROGRAM C
PROGRAM D

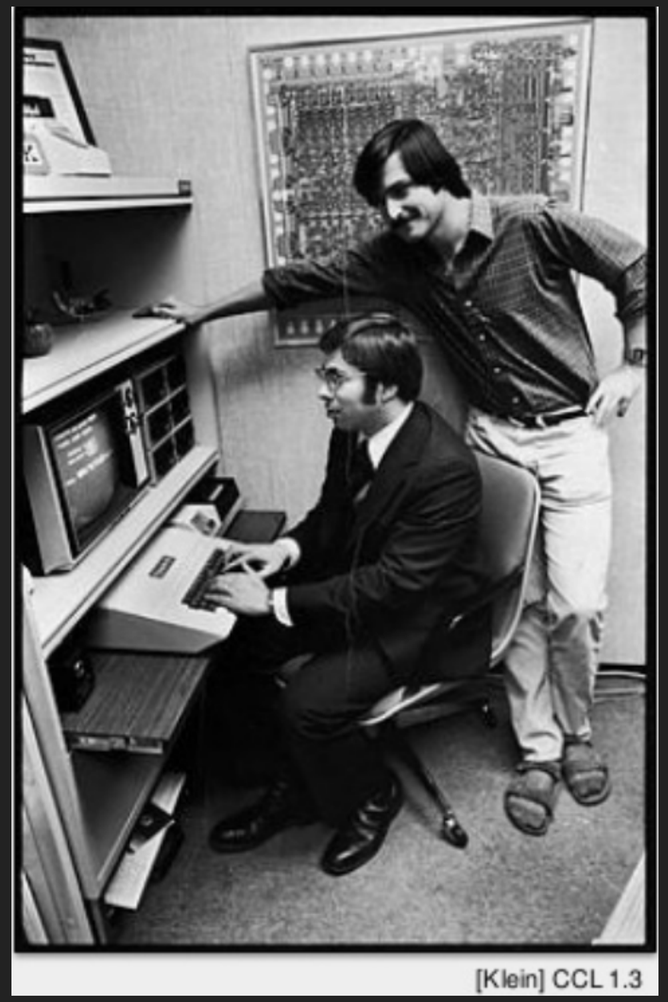
ENVIRONMENTS – NOT SO SIMPLE TO RUN RANDOM PROGRAMS

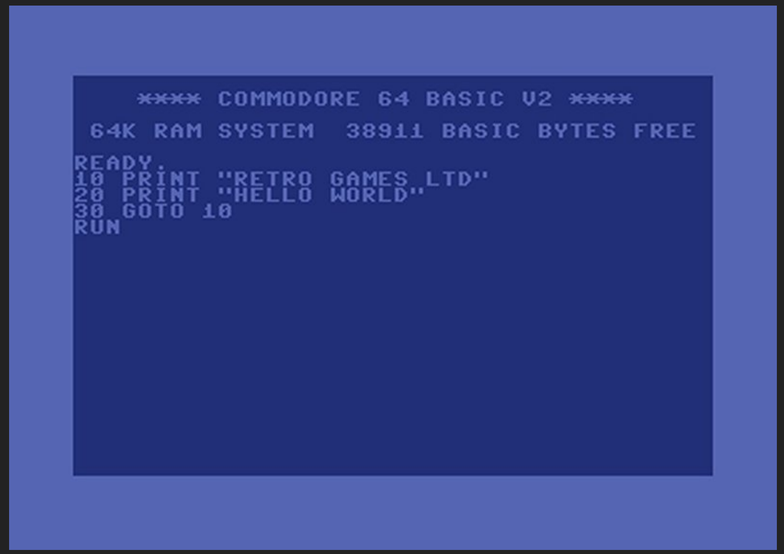
- EMULATORS (INTEL 8080, MOS 6502)
- VIRTUAL MACHINES
- FPGA (RISC-V)
- CUSTOM HW (CEREBRAS CS-1)

テキスト

8-bit cpu, 64kB address space, 96-bit register file, 200 lines of code

6502





https://github.com/krocki/6502

NV6502

```
__device__ void _cp (_6502 *n, u8 _a, u8 _b) { u8 r=_a-_b; C=(_a>=_b); ZN(r); }
__device__ void _ora(_6502 *n) { LDM; A|=d; ZN(A); } //   "OR" Memory with Accumulator
__device__ void _and(_6502 *n) { LDM; A&=d; ZN(A); } //   "AND" Memory with Accumulator
__device__ void _eor(_6502 *n) { LDM; A^=d; ZN(A); } //   "XOR" Memory with Accumulator
__device__ void _cmp(_6502 *n) { LDM; _cp(n,A,d); } //   Compare Memory and Accumulator
__device__ void _cpx(_6502 *n) { LDM; _cp(n,X,d); } //   Compare Memory and Index X
__device__ void _cpy(_6502 *n) { LDM; _cp(n,Y,d); } //   Compare Memory and Index Y
__device__ void _bcc(_6502 *n) { jr(n,!C); } //   Branch on Carry Clear
__device__ void _bcs(_6502 *n) { jr(n,C);  } //   Branch on Carry Set
__device__ void _beq(_6502 *n) { jr(n,Z);  } //   Branch on Result Zero
__device__ void _bit(_6502 *n) { LDM; S=(d>>7) & 1; V=(d>>6) & 1; Z=(d & A)==0; } // Test Bits in Memory with A
__device__ void _bmi(_6502 *n) { jr(n, S);  } //   Branch on Result Minus
__device__ void _bne(_6502 *n) { jr(n,!Z); } //   Branch on Result not Zero
__device__ void _bpl(_6502 *n) { jr(n,!S); } //   Branch on Result Plus
__device__ void _brk(_6502 *n) { B=1;    } //   Force Break
__device__ void _bvc(_6502 *n) { jr(n,!V); } //   Branch on Overflow Clear
__device__ void _bvs(_6502 *n) { jr(n, V);  } //   Branch on Overflow Set
__device__ void _clc(_6502 *n) { C=0; } //   Clear Carry Flag
__device__ void _cld(_6502 *n) { D=0; } //   Clear Decimal Mode
__device__ void _cli(_6502 *n) { I=0; } //   Clear interrupt Disable Bit
__device__ void _clv(_6502 *n) { V=0; } //   Clear Overflow Flag
__device__ void _dec(_6502 *n) { u16 d0 = d; LDM; d--; d &= 0xff; ZN(d); w8(n,d0,d); } //   Decrement Memory by One
__device__ void _dex(_6502 *n) { X--; ZN(X); } //   Decrement Index X by One
__device__ void _dey(_6502 *n) { Y--; ZN(Y); } //   Decrement Index Y by One
__device__ void _inc(_6502 *n) { u16 d0=d; LDM; d++; d &= 0xff; ZN(d); w8(n,d0,d); d=d0; } // Incr Memory by One
__device__ void _inx(_6502 *n) { X++; ZN(X); } //   Increment Index X by One
__device__ void _iny(_6502 *n) { Y++; ZN(Y); } //   Increment Index Y by One
__device__ void _jmp(_6502 *n) { PC=d;} //   Jump to New Location
__device__ void _jsr(_6502 *n) { push16(n,PC-1); PC=d; } //   Jump to New Location Saving Return Address
__device__ void _lda(_6502 *n) { LDM; A=d; ZN(A); } //   Load Accumulator with Memory
__device__ void _ldx(_6502 *n) { LDM; X=d; ZN(X); } //   Load Index X with Memory
__device__ void _ldy(_6502 *n) { LDM; Y=d; ZN(Y); } //   Load Index Y with Memory
__device__ void _lsr(_6502 *n) { LD_A_OR_M(); C=w & 1; w>>=1; ZN(w); ST_A_OR_M(); } // Shift Right One Bit
__device__ void _asl(_6502 *n) { LD_A_OR_M(); C=(w>>7) & 1; w<<=1; ZN(w); ST_A_OR_M();} // Shift Left One Bit
__device__ void _rol(_6502 *n) { LD_A_OR_M(); u8 c = C; C=(w>>7) & 1; w=(w<<1) | c; ZN(w); ST_A_OR_M(); } // Rotate One Bit Left (Memory or Accumulator)
__device__ void _ror(_6502 *n) { LD_A_OR_M(); u8 c = C; C=(w & 1); w=(w>>1) | (c<<7); ZN(w); ST_A_OR_M(); } //   Rotate One Bit Right (Memory or Accumulator)
__device__ void _nop(_6502 *n) { /* No Operation */ }
__device__ void _pha(_6502 *n) { push8(n, A); } //   Push Accumulator on Stack
__device__ void _php(_6502 *n) { push8(n, P | 0x10); } //   Push Processor Status on Stack
__device__ void _pla(_6502 *n) { A=pop8(n); Z=(A==0); S=(A>>7)&0x1;} //   Pull Accumulator from Stack
__device__ void _plp(_6502 *n) { P=pop8(n) & 0xef | 0x20;  } //   Pull Processor Status from Stack
__device__ void _rti(_6502 *n) { P=(pop8(n) & 0xef) | 0x20; PC=pop16(n); } //   Return from Interrupt
__device__ void _rts(_6502 *n) { PC=pop16(n)+1;} //   Return from Subroutine
__device__ void _sec(_6502 *n) { C=1;} //   Set Carry Flag
__device__ void _sed(_6502 *n) { D=1;} //   Set Decimal Mode
__device__ void _sei(_6502 *n) { I=1;} //   Set Interrupt Disable Status
__device__ void _sta(_6502 *n) { w8(n,d,A);} //   Store Accumulator in Memory
__device__ void _stx(_6502 *n) { w8(n,d,X);} //   Store Index X in Memory
__device__ void _sty(_6502 *n) { w8(n,d,Y);} //   Store Index Y in Memory
__device__ void _tax(_6502 *n) { X=A; ZN(X); } //   Transfer Accumulator to Index X
__device__ void _tay(_6502 *n) { Y=A; ZN(Y); } //   Transfer Accumulator to Index Y
__device__ void _tsx(_6502 *n) { X=SP;ZN(X); } //   Transfer Stack Pointer to Index X
__device__ void _txa(_6502 *n) { A=X; ZN(A); } //   Transfer Index X to Accumulator
__device__ void _txs(_6502 *n) { SP=X; } //   Transfer Index X to Stack Pointer
__device__ void _tya(_6502 *n) { A=Y; ZN(A); } //   Transfer Index Y to Accumulator
```

~100.000 programs/second

1 run: 1M clock cycles

https://github.com/krocki/nv6502

NV6502

GPU

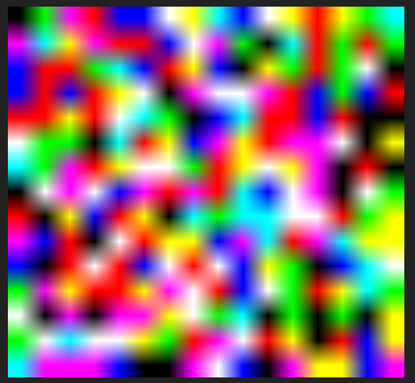
~100.000 programs/second

1 run: 1M clock cycles

TEST thousands of programs at the same time

https://github.com/krocki/nv6502

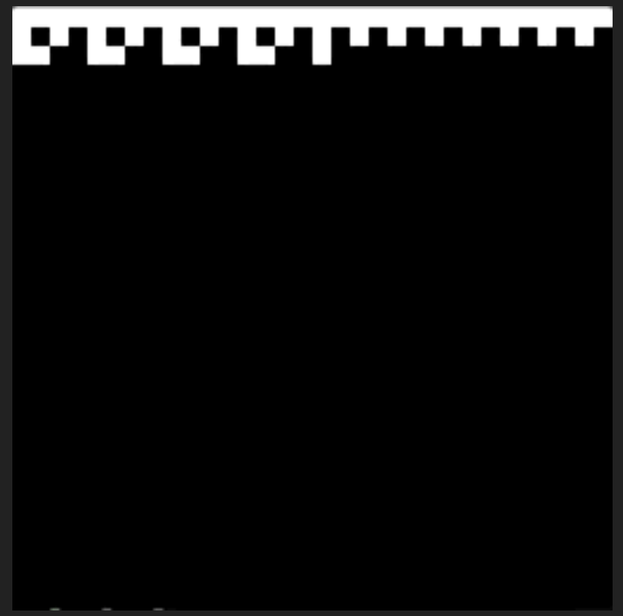
PROGRAM



AN EXAMPLE USING 6502 CPU

```
a9e1 8500 a901 8501 a020 a200 4100 9100
e600 d0f6 e601 a601 e006 d0ee 60
```

29 BYTES

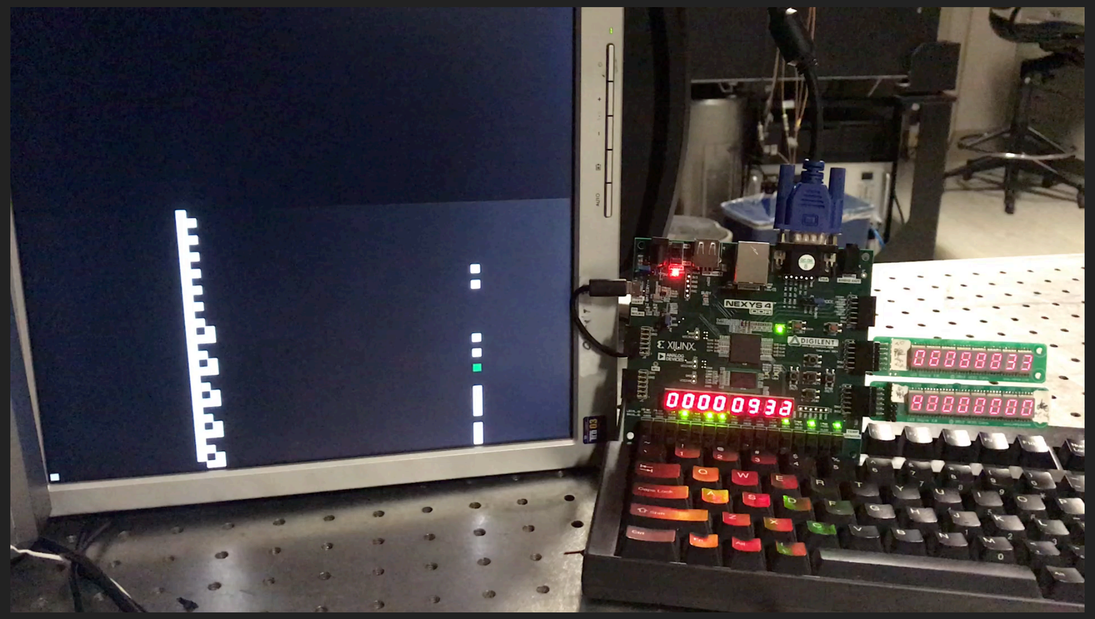


VISIBLE/EXTERNAL MEMORY

AN EXAMPLE USING 6502 CPU

```
a9e1 8500 a901 8501 a020 a200 4100 9100
e600 d0f6 e601 a601 e006 d0ee 60
```

29 BYTES

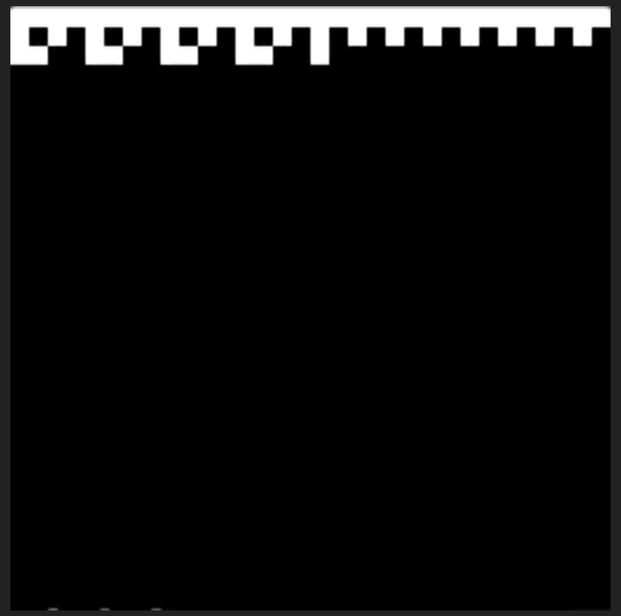


```
start:
  lda #$e1
  sta $0
  lda #$01
  sta $1
  ldy #$20

write:
  ldx #$00
  eor ($0, x)
  sta ($0),y

  inc $0
  bne write
  inc $1
  ldx $1
  cpx #$06
  bne write

  rts
```



VISIBLE/EXTERNAL MEMORY

OBSERVATIONS

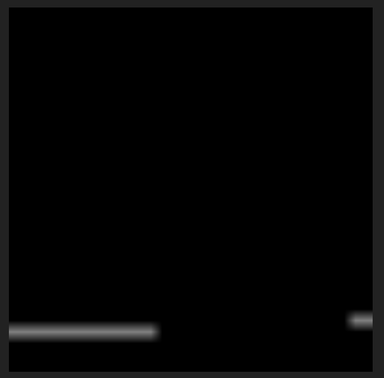
- Challenges: constraining the solution space (i.e. enforcing subroutines), learning recursion is hard using current approaches, multi-agent cooperation
- Works for simple tasks:
  - a) copy, b) repeat-copy, c) reverse-copy
  - d) basic sort, e) filter

http://olab.is.s.u-tokyo.ac.jp/~kamil.rocki/dca/

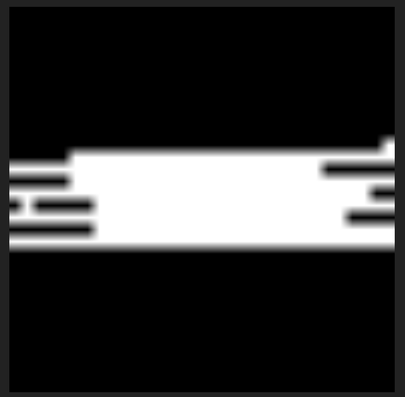
LEARNED BEHAVIORS

FILL

COUNTER



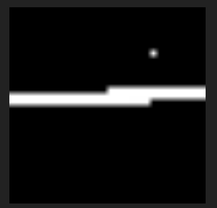
SHAPE BY EROSION



RECURSIVE

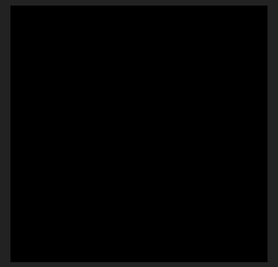
SELF-MODIFICATION

QUICKFILL



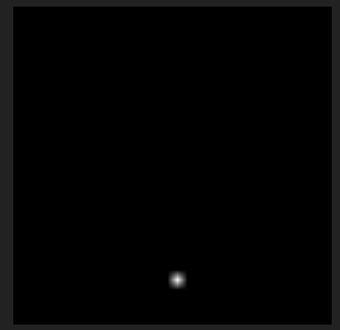
ADDITIVE FILL

SUBROUTINES



LOOPS

POINT-LINE-BLOCK

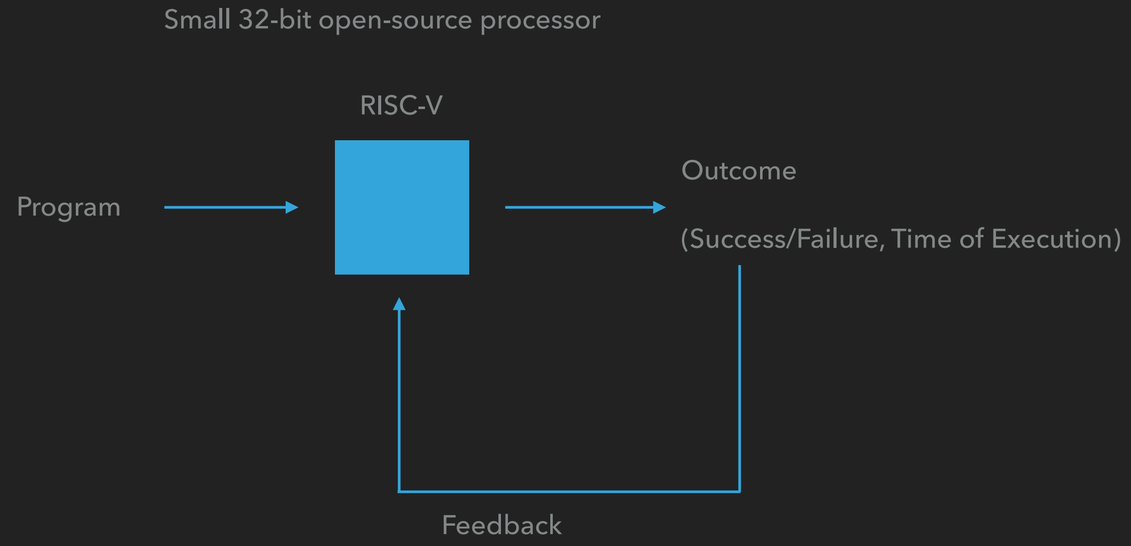


RAM USAGE / CODE GENERATION

LOOP WITH ‘IF’

RL-BASED ATTEMPTS

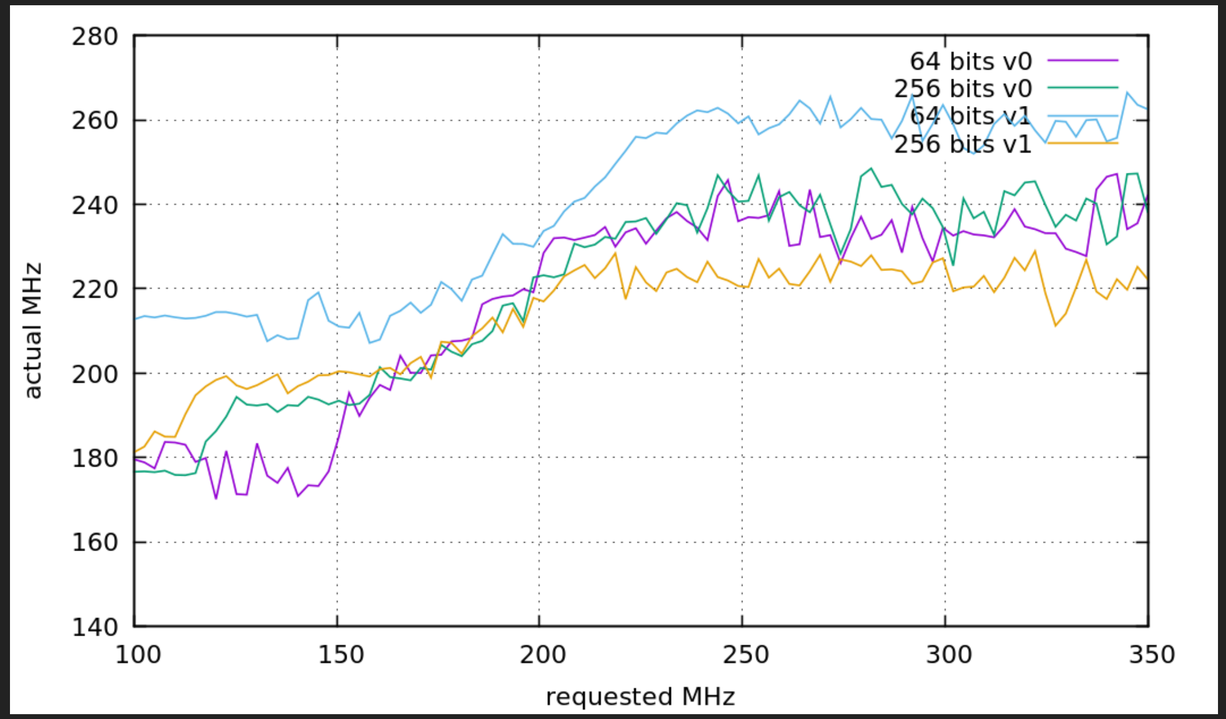
In an FPGA



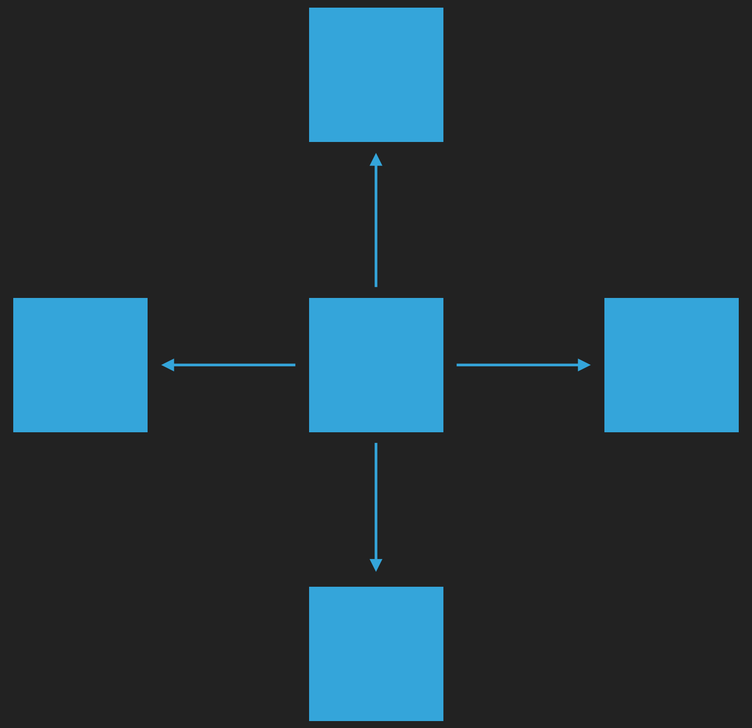
160

28nm ZYNQ Z-7045

LEARNING TO MEMCPY DATA USING A SINGLE CORE



LEARNING DISTRIBUTED COMPUTATION



AUGMENT THE ACTION SPACE WITH SEND/RECEIVE

DEFINE A GLOBAL OBJECTIVE (i.e. SORT, MAX)