COEN 210 DESIGN PROJECT

Improvement in Performance via Parallelism

Gousia Sultana Syeda
Alex Whelan

RISC-V Assembly Code

```
# int A[] = {1,2,5,9};
# x10 -> base array, x11 -> size, x12 -> key
# x5 : index
# x6 : value
asm_find:
   lw x6, 0(x10)              # v = A[i]
   addi x10, x10, 4           # increment A[i+1]
   addi x5, x5, 1             # i++
   beq x6, x12, found         # if(A[i] == key) then return
   beq x5, x11, Exit          # if i == size then Exit
   beq x0, x0, asm_find       # goto top of loop

found:
   add x10, x5, 1          # return index + 1 if found
   jal x1
Exit:
   add x10, x0, x0         # if Not found return 0
   jal x1
```

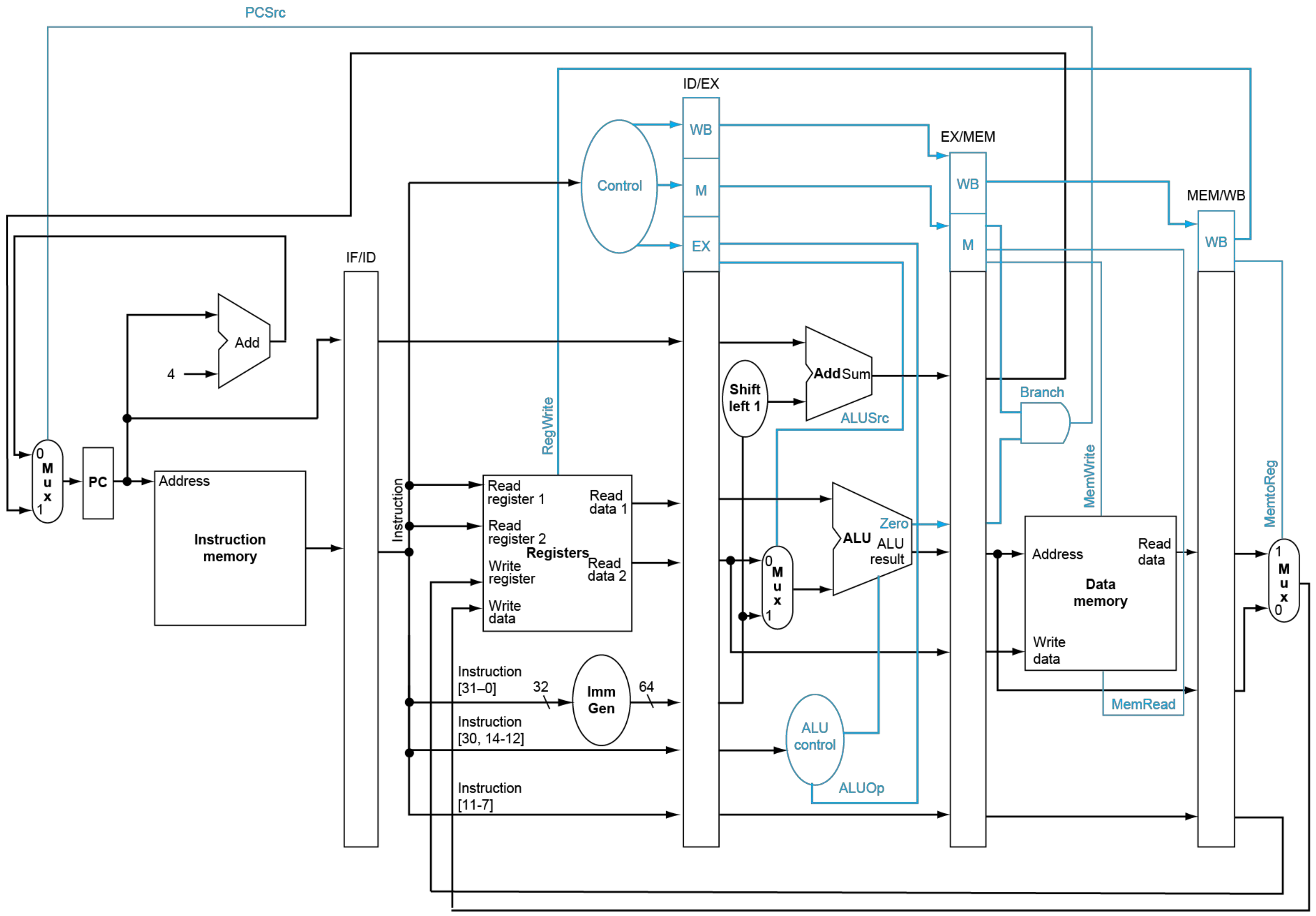


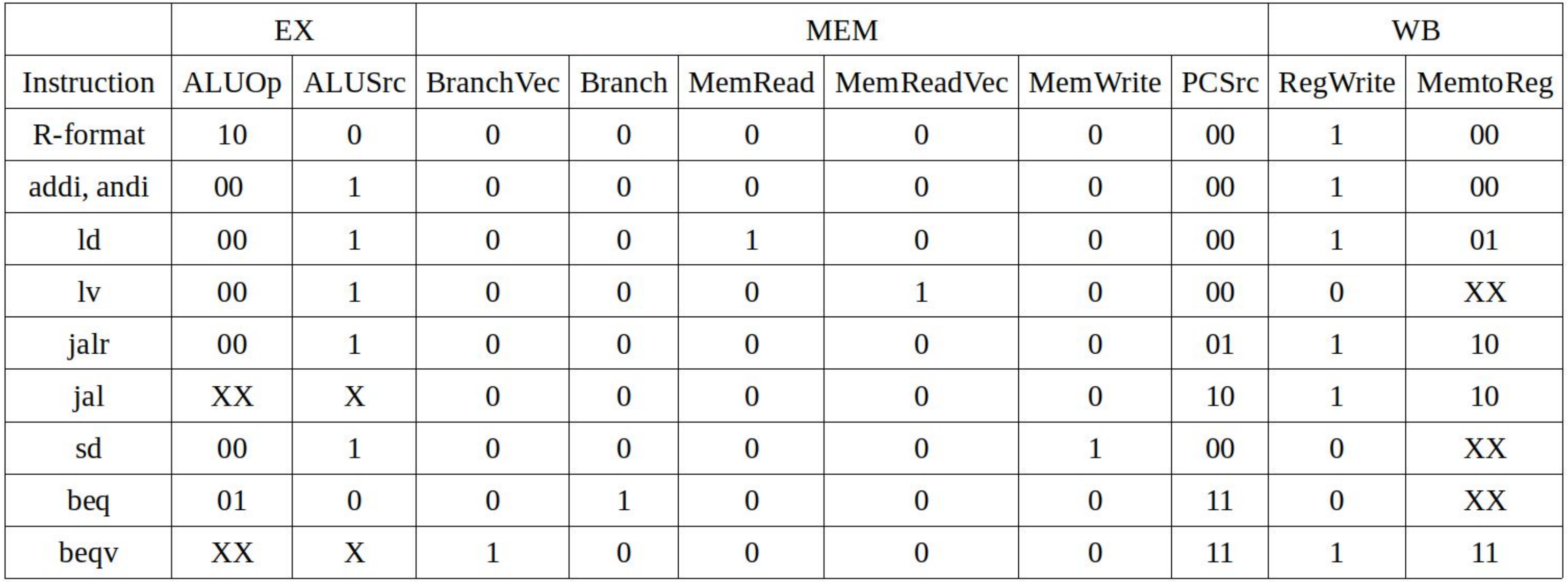
Image: David A. Patterson, John L. Hennessy - Computer Organization and Design The Hardware Software Interface [RISC-V Edition]

Vector Code

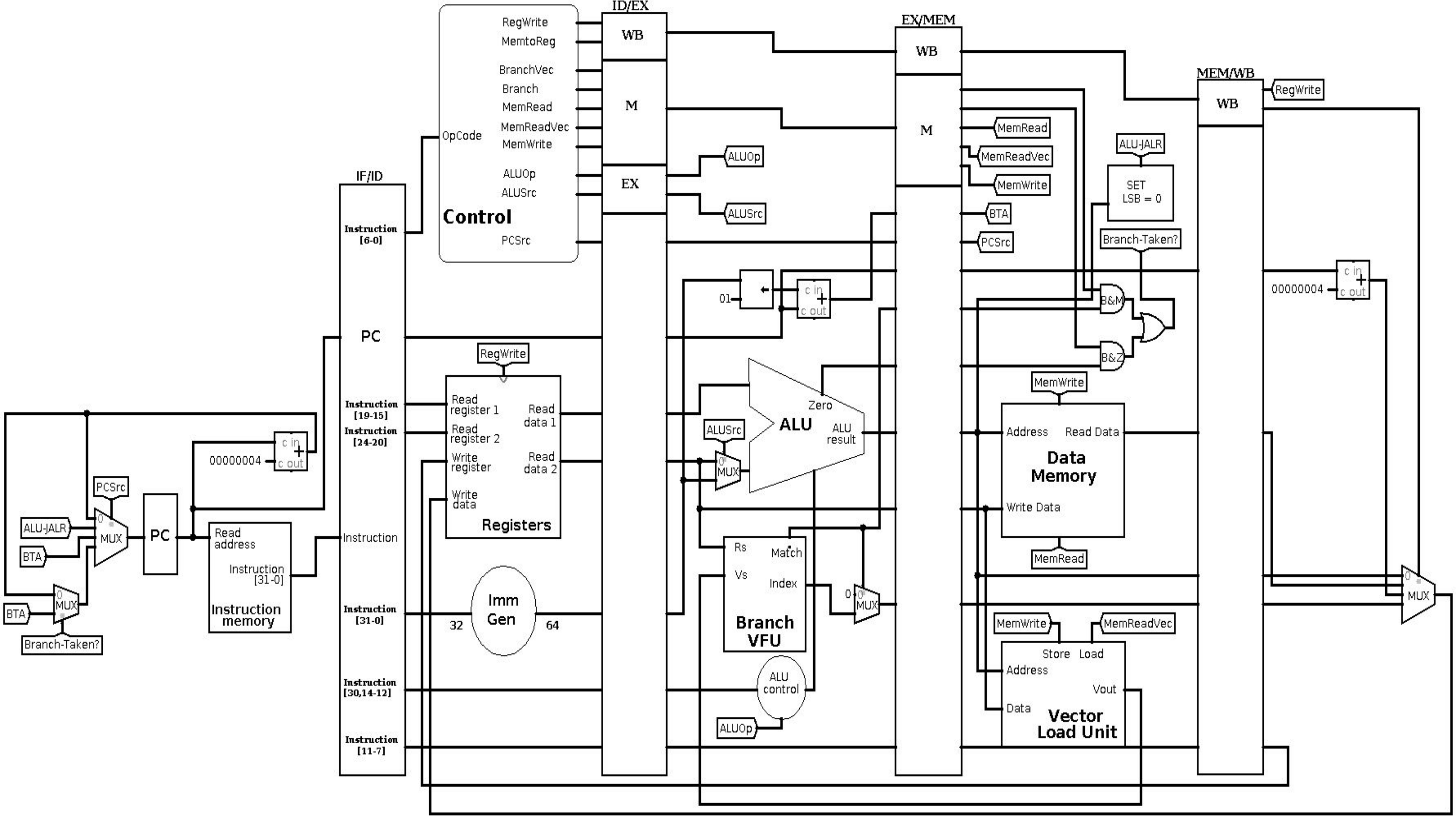
```
# int A[] = {1,2,5,9};
# x10 -> base array, x11 -> size, x12 -> key
# x5 : index
# x6 : value
asm_find:
   lv v1, 0(x10)           # v1 = A[i...i+31]
   addi x10, x10, 128      # increment base address A + 128 (4 bytes * 32 elements)
   beqv x127, x6, found    # CMP(membership) key == v1 (v1 is implicit), store match index or 0 in x127
   addi x5, x5, 32         # i += 32
   bge x5, x11, Exit       # if i >= size then Exit
   beq x0, x0, asm_find    # goto top of loop

found:
   add x10, x5, x127      # move to result iteration(0,32,64,...n) + found index
   addi x10, x10, 1       # return index + 1 if found
   jal x0, x1
Exit:
   add x10, x0, x0        # if Not found return 0
   jal x0, x1
```

Datapath and Control

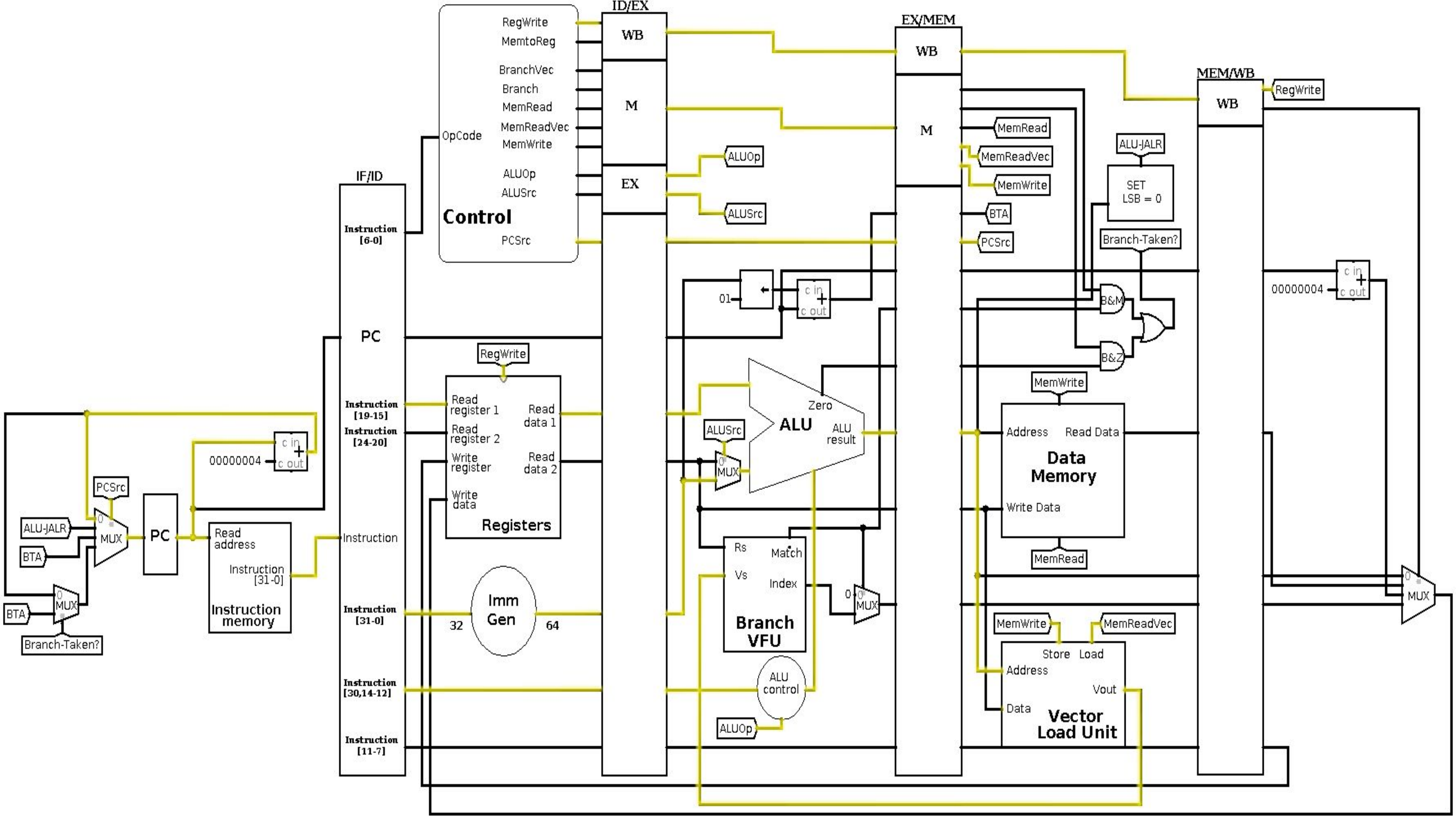


	EX		MEM						WB	
Instruction	ALUOp	ALUSrc	BranchVec	Branch	MemRead	MemReadVec	MemWrite	PCSrc	RegWrite	MemtoReg
R-format	10	0	0	0	0	0	0	00	1	00
addi, andi	00	1	0	0	0	0	0	00	1	00
ld	00	1	0	0	1	0	0	00	1	01
lv	00	1	0	0	0	1	0	00	0	XX
jalr	00	1	0	0	0	0	0	01	1	10
jal	XX	X	0	0	0	0	0	10	1	10
sd	00	1	0	0	0	0	1	00	0	XX
beq	01	0	0	1	0	0	0	11	0	XX
beqv	XX	X	1	0	0	0	0	11	1	11

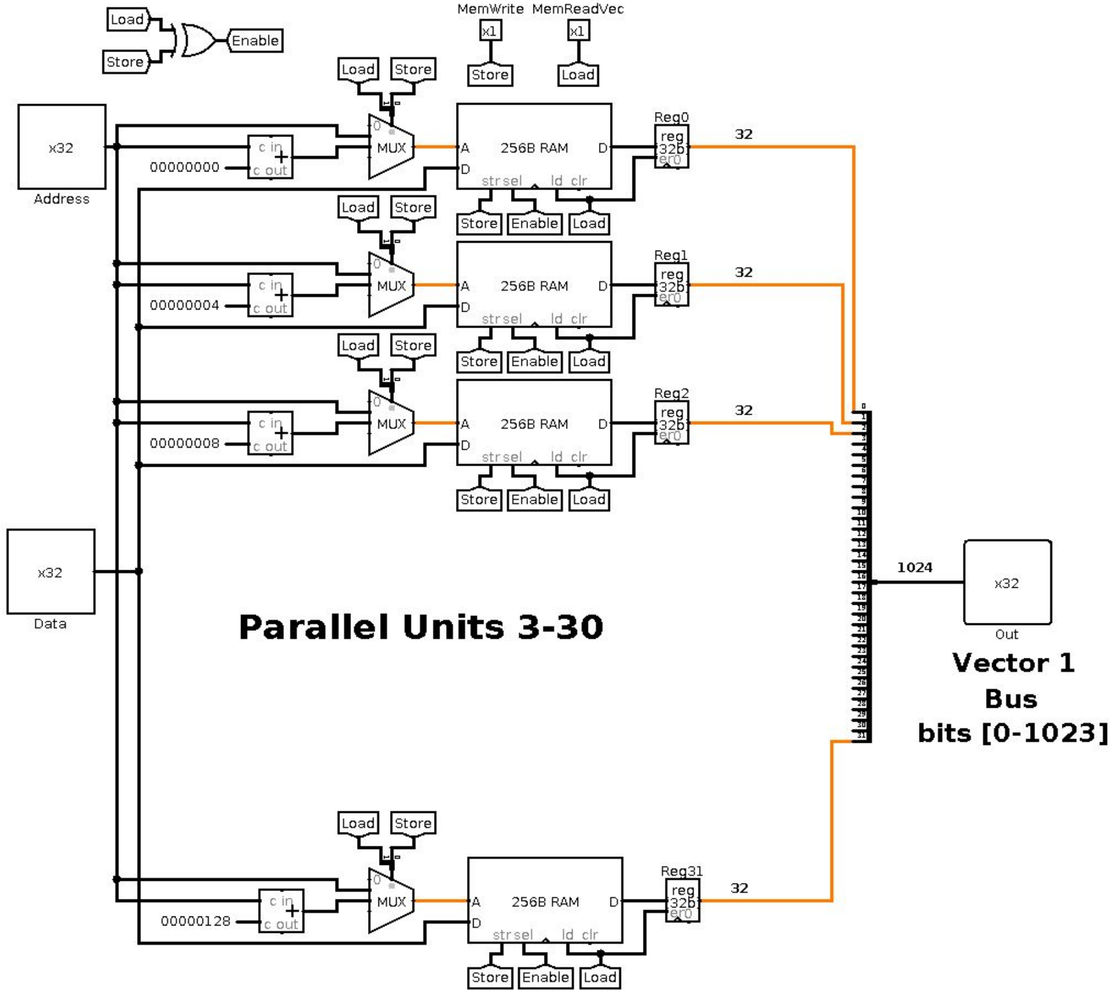

ID/EX
EX/MEM
MEM/WB
IF/ID
Control
RegWrite
MemtoReg
BranchVec
Branch
MemRead
MemReadVec
MemWrite
ALUOp
ALUSrc
OpCode
PCSrc
WB
M
EX
Instruction [6-0]
PC
Instruction [19-15]
Instruction [24-20]
Instruction
Instruction [31-0]
Instruction [30,14-12]
Instruction [11-7]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Imm Gen
32
64
ALU
Zero
ALU result
MUX
ALU control
Branch VFU
Rs
Vs
Match
Index
01
c in
c out
00000004
Read address
Instruction [31-0]
Instruction memory
ALU-JALR
BTA
Branch-Taken?
SET LSB = 0
B&M
B&Z
Data Memory
Address
Read Data
Write Data
Vector Load Unit
Store
Load
Data
Vout
0

Vector Load Unit

Immediate Format
lv vd, offset(rs1)



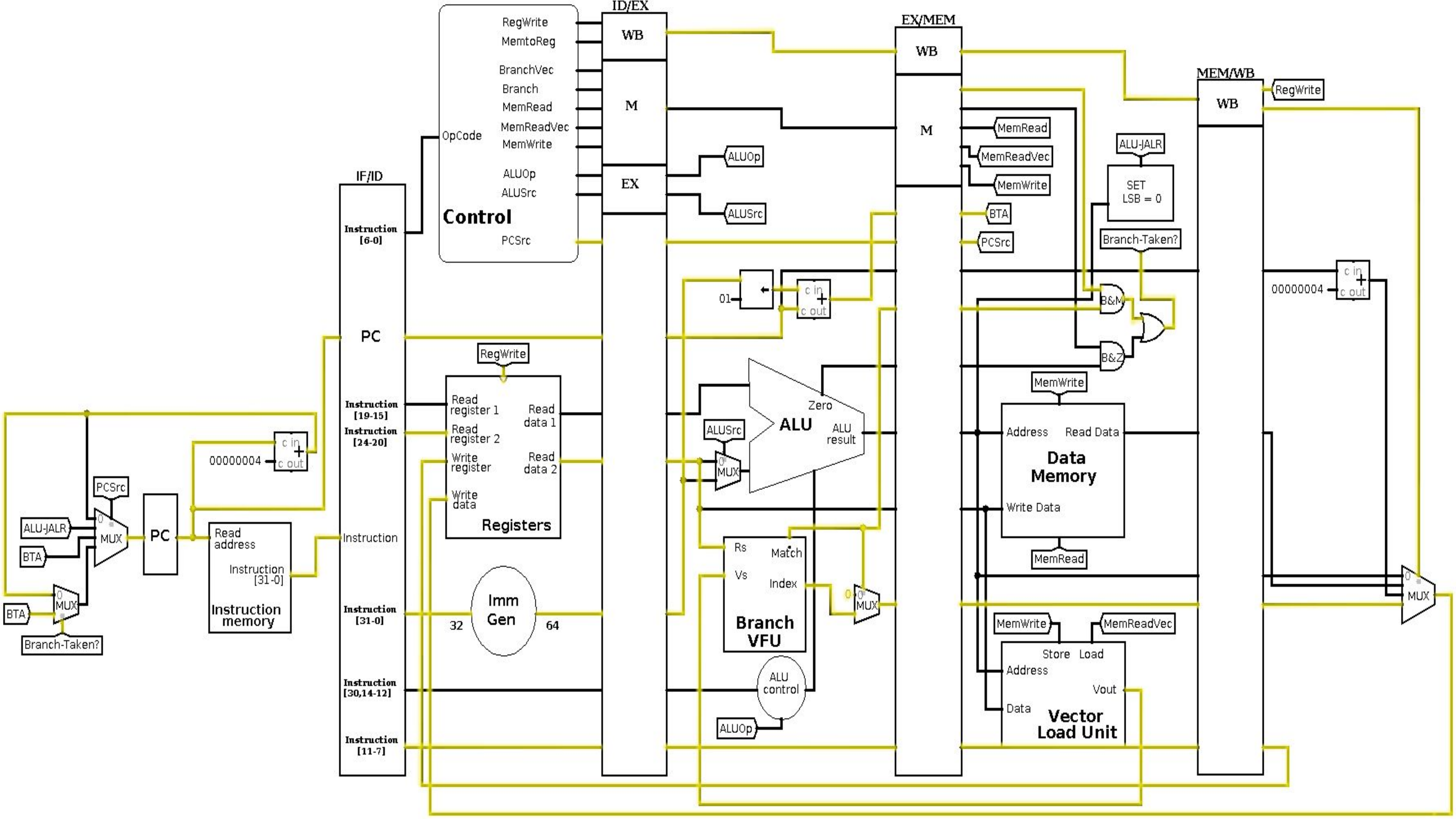
ID/EX
EX/MEM
MEM/WB
IF/ID
RegWrite
MemtoReg
BranchVec
Branch
MemRead
MemReadVec
MemWrite
ALUOp
ALUSrc
OpCode
Control
PCSrc
WB
M
EX
Instruction [6-0]
PC
Instruction [19-15]
Instruction [24-20]
Instruction
Instruction [31-0]
Instruction [30,14-12]
Instruction [11-7]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Imm Gen
32
64
ALU
Zero
ALU result
MUX
Branch VFU
Rs
Vs
Match
Index
ALU control
01
0
Data Memory
Address
Read Data
Write Data
Vector Load Unit
Store
Load
Data
Vout
SET LSB = 0
ALU-JALR
Branch-Taken?
B&M
B&Z
BTA
00000004
c in
c out
Read address
Instruction memory



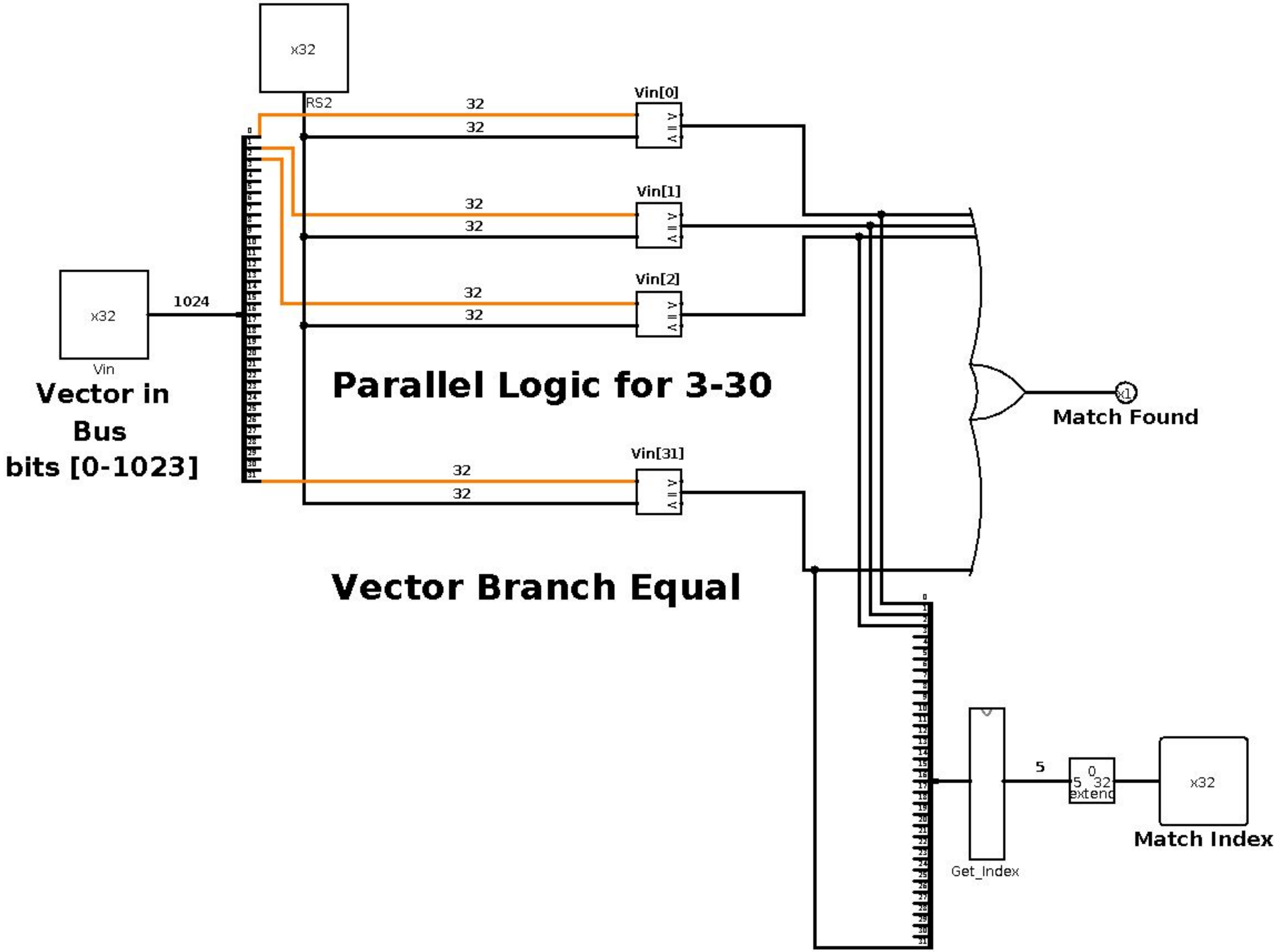
Vector Load Unit

Branch Vector Functional Unit

SB Format
beqv rs1, rs2, branch-target-address



ID/EX
EX/MEM
MEM/WB
IF/ID
RegWrite
MemtoReg
BranchVec
Branch
MemRead
MemReadVec
MemWrite
ALUOp
ALUSrc
OpCode
Control
PCSrc
WB
M
EX
Instruction [6-0]
PC
Instruction [19-15]
Instruction [24-20]
Instruction
Instruction [31-0]
Instruction [30,14-12]
Instruction [11-7]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Imm Gen
32
64
Read address
Instruction [31-0]
Instruction memory
00000004
c in
c out
MUX
ALU-JALR
BTA
Branch-Taken?
01
ALU
Zero
ALU result
Rs
Vs
Match
Index
Branch VFU
ALU control
SET LSB = 0
B&M
B&Z
Address
Read Data
Data Memory
Write Data
Store
Load
Data
Vout
Vector Load Unit
0



Vector Branch Equal

Performance Analysis

Clock Cycle Time

	Baseline	**Improved (with Vector Unit)**
IF STAGE	305ps	330ps
ID STAGE	150ps	150ps
EX STAGE	275ps	225ps
MEM STAGE	250ps	450ps
WB STAGE	25ps	195ps
Clock Cycle TIme	**305ps**	**450ps**

Instruction count in terms of n

Baseline pipeline implementation:

$5n + (n-1) + 2 = 6n + 1$

Improved pipeline implementation:

$5(n/32) + ((n/32) - 1) + 2 = (6n/32) + 1$

Total Cycles

Minimum Cycles without stalls: $n + k - 1$

n = number of instructions

k = number of stages in the pipeline = 5

Baseline = $6n + 1 + 5 - 1 = 6n + 5$ cycles

Optimized = $(6n/32) + 1 + 5 - 1 = (6n/32) + 5$ cycles

CPI

Global CPI = total cycles / instruction count

Baseline = $(6n + 5) / (6n + 1) = 1$ as $n \to \infty$

Optimized = $((6n/32) + 5) / ((6n/32) + 1) = 1$ as $n \to \infty$

CPI = 1 which is consistent with pipeline implementation

Execution Time

Execution Time = IC * CPI * Cycle Time = IC * (total cycles/IC) * Cycle Time
Execution Time = total cycles * cycle time

Execution time baseline = (6n + 5) * 305
Execution time optimized = ((6n/32) + 5) * 450

$$\lim_{n->\infty} \frac{(6n+5)*305}{(\frac{6n}{32}+5)*450} \approx 21.6889$$

Approximately **21.7** times the speedup!