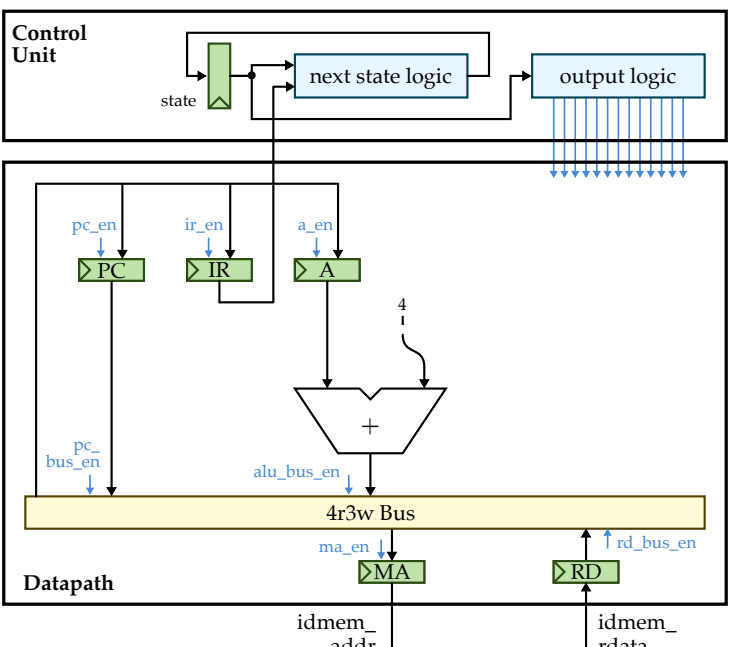
# 3. 64-State Multi-Cycle Processor Microarchitecture

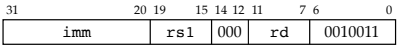
| state | pc bus en | alu bus en | rd bus en | pc en | ir en | a en | ma en | mem val |
|---|---|---|---|---|---|---|---|---|
| F0 | | | | | | | | |
| F1 | | | | | | | | |
| F2 | | | | | | | | |



**ADDI**

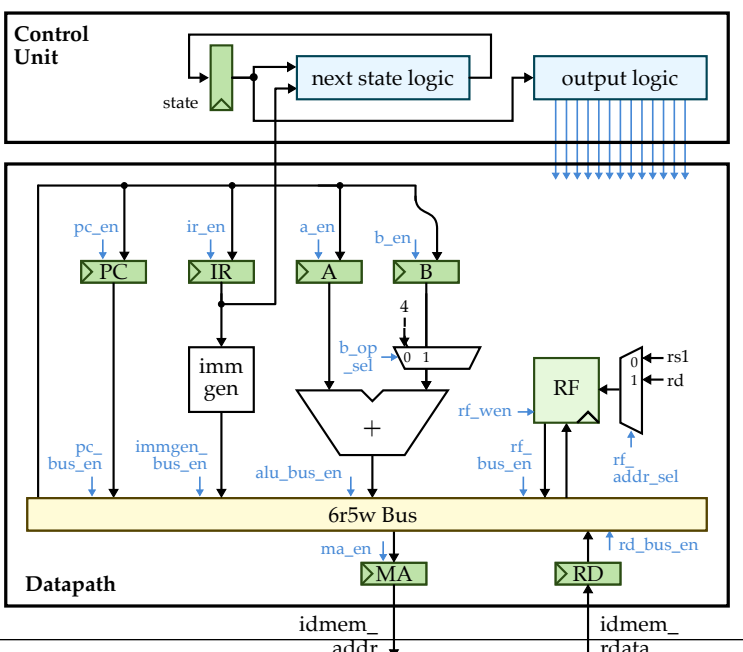
```
addi rd, rs1, imm
```

R[rd] ← R[rs1] + sext(imm)
PC ← PC + 4




I-type Immediate

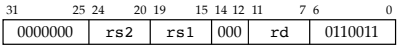
| state | pc bus en | imm bus en | alu bus en | rf bus en | rd bus en | pc en | ir en | a en | b en | bop sel | rf addr sel | rf wen | ma en | mem val |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F0 | 1 | | 0 | | 0 | 0 | 0 | 1 | | | | | 1 | 0 |
| F1 | 0 | | 1 | | 0 | 1 | 0 | 0 | | | | | 0 | 1 |
| F2 | 0 | | 0 | | 1 | 0 | 1 | 0 | | | | | 0 | 0 |
| AI0 | | | | | | | | | | | | | | |
| AI1 | | | | | | | | | | | | | | |
| AI2 | | | | | | | | | | | | | | |



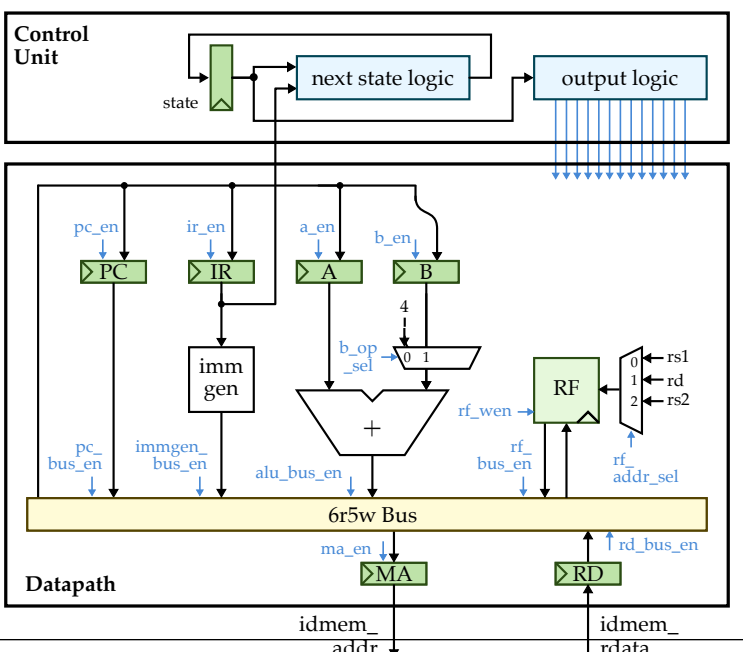
**ADD**

```
add rd, rs1, rs2
```

R[rd] ← R[rs1] + R[rs2]
PC ← PC + 4




| state | pc bus en | imm bus en | alu bus en | rf bus en | rd bus en | pc en | ir en | a en | b en | bop sel | rf addr sel | rf wen | ma en | mem val |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | x | x | 0 | 1 | 0 |
| F1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | x | 0 | 0 | 1 |
| F2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | x | x | 0 | 0 | 0 |
| A0 | | | | | | | | | | | | | | |
| A1 | | | | | | | | | | | | | | |
| A2 | | | | | | | | | | | | | | |



## Control Signal Pseudo-Code

### Fetch States

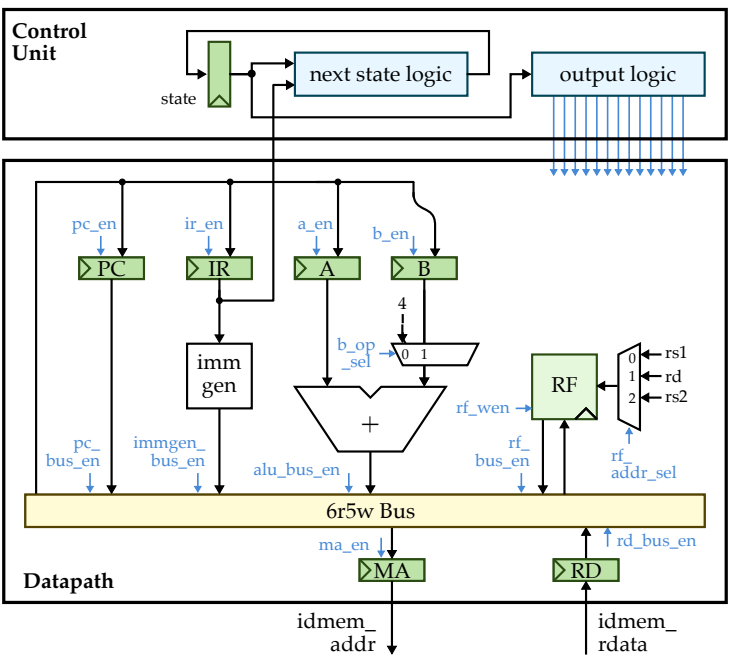
F0: MA ← PC; A ← PC;
F1: RD ← M[MA]; PC ← A+4
F2: IR ← RD; goto inst

### `addi rd, rs1, imm`

AI0: A ← R[`rs1`]
AI1: B ← sext(`imm_i`)
AI2: R[`rd`] ← A + B; goto F0

### `add rd, rs1, rs2`

A0: A ← R[`rs1`]
A1: B ← R[`rs2`]
A2: R[`rd`] ← A + B; goto F0

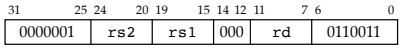


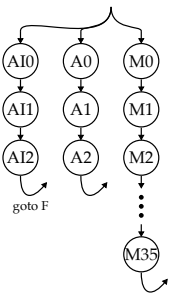
**MUL**

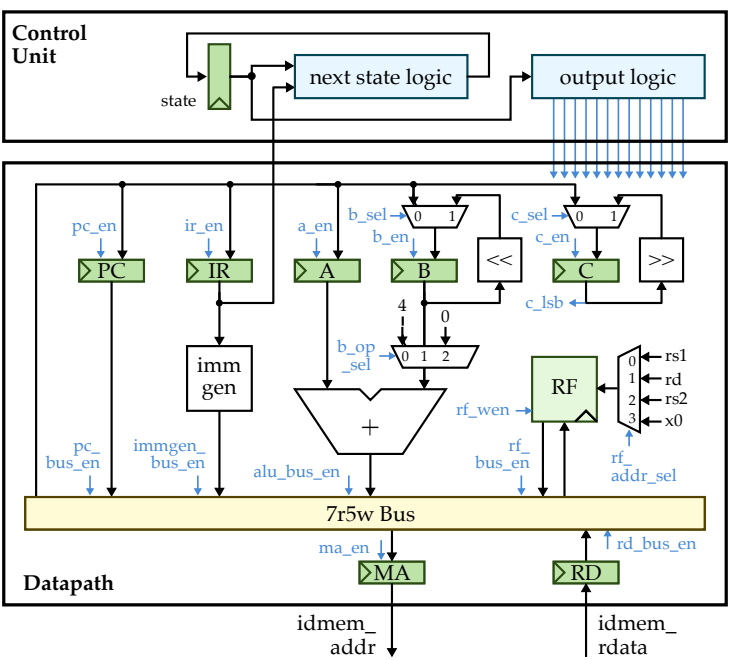
```
mul rd, rs1, rs2
```

R[rd] ← R[rs1] × R[rs2]

PC ← PC + 4



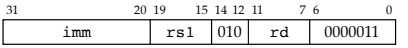



**LW**

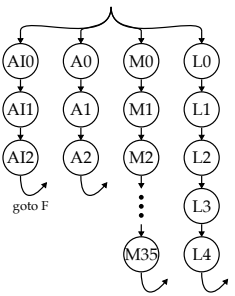
```
lw rd, imm(rs1)
```

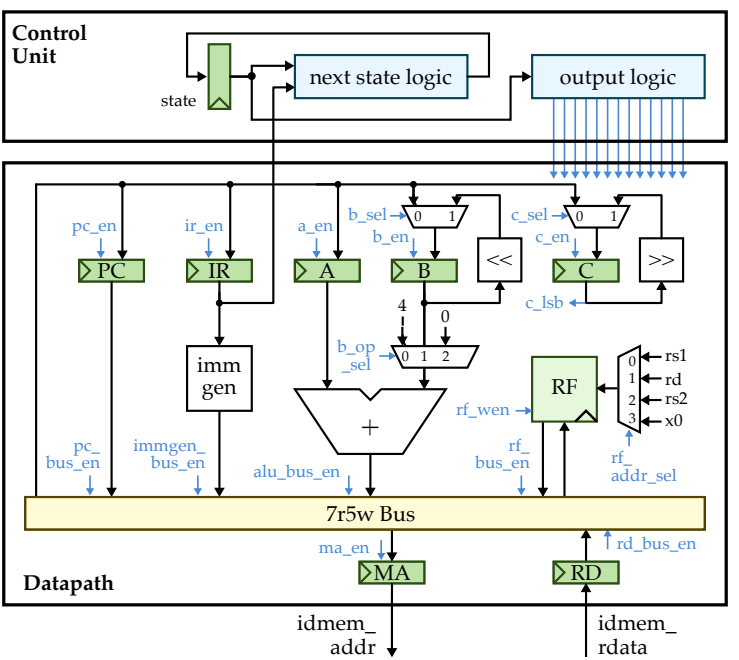
R[rd] ← M[ R[rs1] + sext(imm) ]

PC ← PC + 4

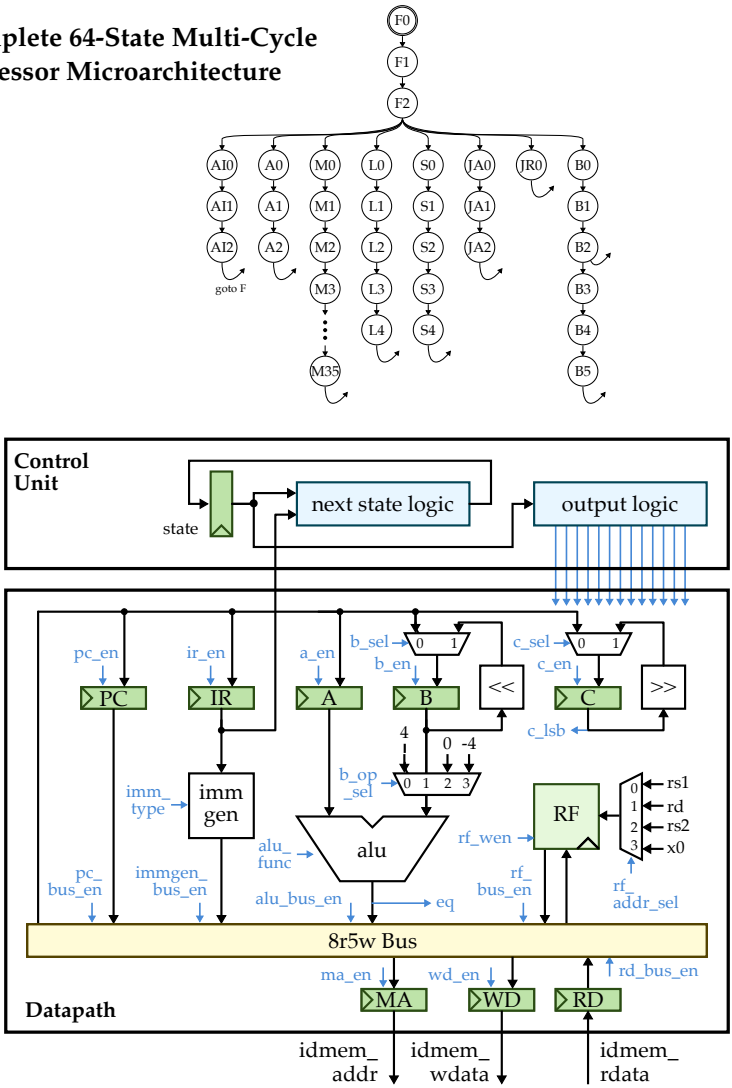



I-type Immediate



## Complete 64-State Multi-Cycle Processor Microarchitecture



### Fetch States

F0: MA ← PC; A ← PC;
F1: RD ← M[MA]; PC ← A+4
F2: IR ← RD; goto inst

### `addi rd, rs1, imm`

AI0: A ← R[`rs1`]
AI1: B ← sext(`imm_i`)
AI2: R[`rd`] ← A + B

### `add rd, rs1, rs2`

A0: A ← R[`rs1`]
A1: B ← R[`rs2`]
A2: R[`rd`] ← A + B

### `mul rd, rs1, rs2`

M0: A ← R[x0]
M1: B ← R[`rs1`]
M2: C ← R[`rs2`]
M3: A ← C[0] ? A + B : A;
B ← B << 1; C ← C >> 1
M4: A ← C[0] ? A + B : A;
B ← B << 1; C ← C >> 1
...
M34: R[`rd`] ← C[0] ? A + B : A;
goto F0

### `lw rd, imm(rs1)`

L0: A ← R[`rs1`]
L1: B ← sext(`imm_i`)
L2: MA ← A + B
L3: RD ← M[MA]
L4: R[`rd`] ← RD; goto F0

### `sw rs2, imm(rs1)`

S0: WD ← R[`rs2`]
S1: A ← R[`rs1`]
S2: B ← sext(`imm_s`)
S3: MA ← A + B
S4: M[MA] ← WD; goto F0

### `jal rd, imm`

JA0: R[`rd`] ← PC
JA1: B ← sext(`imm_j`)
JA2: PC ← A + B; goto F0

### `jr rs1`

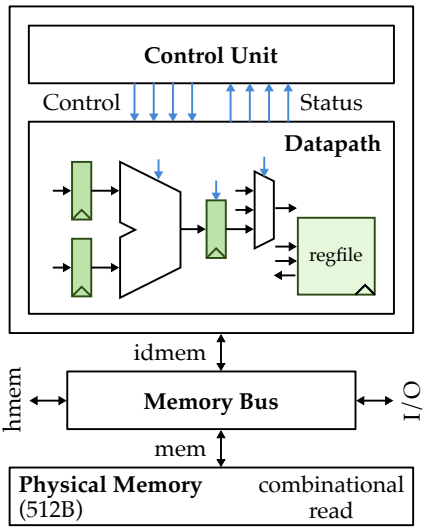
JR0: PC ← R[`rs1`]; goto F0

### `bne rs1, rs2, imm`

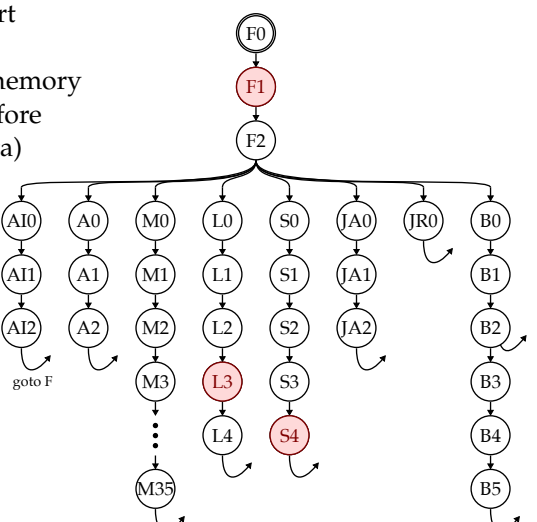
B0: A ← R[`rs1`]
B1: B ← R[`rs2`]
B2: B ← sext(`imm_b`); if A == B goto F0
B3: A ← PC
B4: A ← A − 4
B5: PC ← A + B; goto F0

## System Integration

- Memory-mapped input/output enables multi-cycle processor to external devices by reading/writing memory addresses
- Supporting the memory wait signal simply involves staying in specific states if the memory is waiting
  - Need to integrate memory wait signal into specific control signals creating a Mealy FSM (i.e., reg en, rf wen)




- Now possible to support physical memory with sequential read (send memory read request in state before state requiring read data)



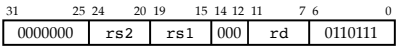
## Adding a Complex Instruction

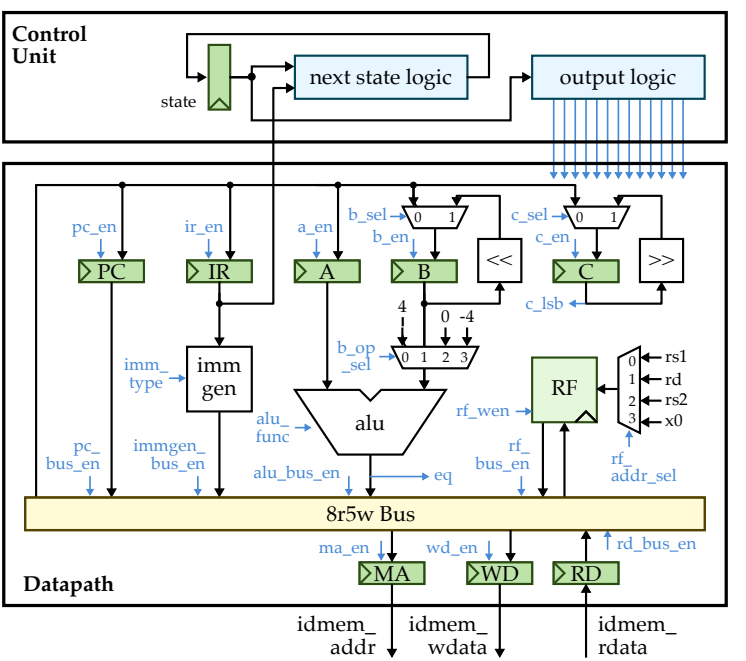
**ADD.MM**

`add.mm rd, rs1, rs2`

M[ R[rd] ] ← M[ R[rs1] + M[ R[rs2] ]

PC ← PC + 4



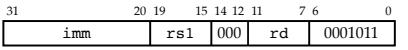



## Adding a New Auto-Incrementing Load Instruction

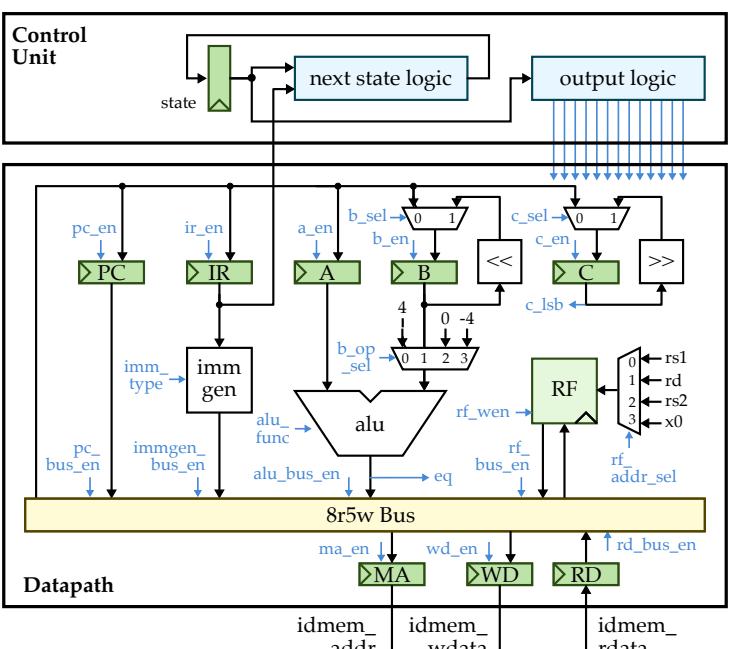
**LW.AI**

`lw.ai rd, imm(rs1)`

R[`rd`] ← M[ R[`rs1`] + sext(`imm`) ]
R[`rs1`] ← R[`rs1`] + 4
PC ← PC + 4




I-type immediate

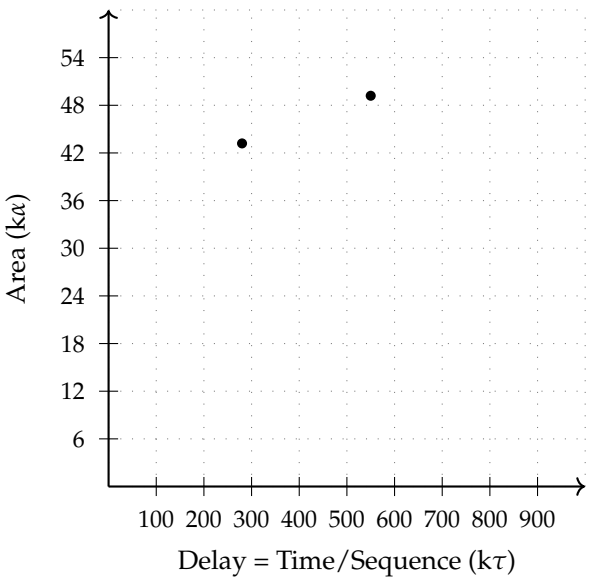


## 3.1. Analyzing Delay and Area

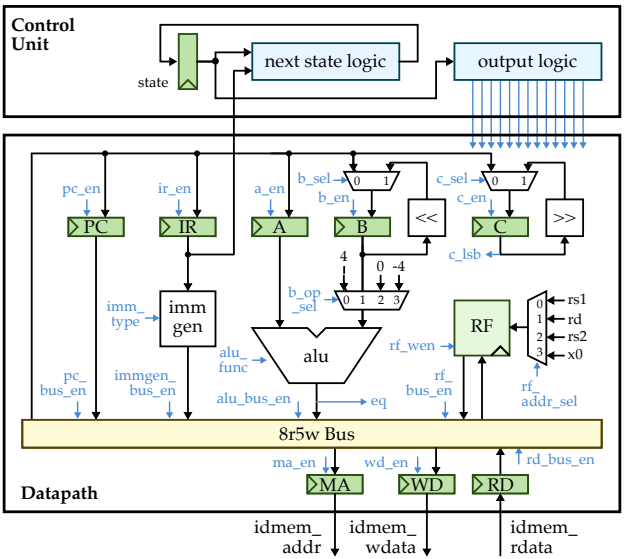
$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Avg Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}} = \frac{\text{Time}}{\text{Program}}$$

**Results for vector-vector-add micro-benchmark**

| µArch | Inst/Prog | Cycles/Inst | Time/Cycle | Time/Prog | Area |
|---|---|---|---|---|---|
| SC | 189 | 1.0 | 1.48 k$\tau$ | 280 k$\tau$ | 43.4 k$\alpha$ |
| MC-15 | 189 | 4.2 | 688 $\tau$ | 546 k$\tau$ | 49.3 k$\alpha$ |
| MC-64 | | | | | |



## Estimating minimum time/cycle ($T_C$) and area




| Component | $t_{pd}$ | Area |
|---|---|---|
| 32b 2-to-1 Mux | 8 $\tau$ | 0.4 k$\alpha$ |
| 32b 4-to-1 Mux | 11 $\tau$ | 0.8 k$\alpha$ |
| 32b ALU | 115 $\tau$ | 2.1 k$\alpha$ |
| ImmGen Unit | 11 $\tau$ | 0.8 k$\alpha$ |
| 32b 8r5w Bus | 30 $\tau$ | 0.4 k$\alpha$ |
| 32b Register ($t_{cq}$) | 9 $\tau$ | 0.9 k$\alpha$ |
| 32b 1r1w Regfile ($t_{cq}$) | 25 $\tau$ | 12.0 k$\alpha$ |
| 32b 1r1w Regfile Read | 25 $\tau$ | |
| 512B Memory ($t_{cq}$) | 650 $\tau$ | |
| 512B Memory Read | 650 $\tau$ | |

| | |
|---|---|
| 32b Register($t_{setup}$) | 10 $\tau$ |
| 32b 1r1w Regfile ($t_{setup}$) | 20 $\tau$ |
| 512B Memory ($t_{setup}$) | 600 $\tau$ |

**Estimating time/program**

How long in units of $\tau$ will it take to execute the vector-vector add program assuming `n` is 21?

Pseudo-Code

```
for i in range(n):
  dest[i] = src0[i] + src1[i]
```

Assembly Code

```
 # assume x1 holds address of dest[i]
 # assume x2 holds address of src0[i]
 # assume x3 holds address of src1[i]
 # assume x4 holds n (counting down)
loop:
 lw    x5, 0(x1)
 lw    x6, 0(x2)
 add   x7, x5, x6
 sw    x7, 0(x3)
 addi  x1, x1, 4
 addi  x2, x2, 4
 addi  x3, x3, 4
 addi  x4, x4, -1
 bne   x4, x0, loop
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `lw x5, 0(x1)` | | | | | | | | | | | | | | | | | | | | |
| `lw x6, 0(x2)` | | | | | | | | | | | | | | | | | | | | |
| `add x7, x5, x6` | | | | | | | | | | | | | | | | | | | | |

| | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `sw x7, 0(x3)` | | | | | | | | | | | | | | | | | | | |
| `addi x1, x1, 4` | | | | | | | | | | | | | | | | | | | |
| `addi x2, x2, 4` | | | | | | | | | | | | | | | | | | | |

| | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `addi x3, x3, 4` | | | | | | | | | | | | | | | | | | | | | |
| `addi x4, x4, -1` | | | | | | | | | | | | | | | | | | | | | |
| `bne x4, x0, loop` | | | | | | | | | | | | | | | | | | | | | |