Modern Processor Design (V): All at Once

Hung-Wei Tseng

Recap: Missing opportunities

```
for(i = 0; i < count; i++) {
    int64_t temp = a[i];
    a[i] = b[i];
    b[i] = temp;
}
```

```
:
movq    (%rcx,%rax), %r8
movq    (%rdi,%rax), %rsi
movq    %r8, (%rdi,%rax)
movq    %rsi, (%rcx,%rax)
addq    $8, %rax
cmpq    %r9, %rax
jne     .L9
movq    (%rcx,%rax), %r8
movq    (%rdi,%rax), %rsi
movq    %r8, (%rdi,%rax)
movq    %rsi, (%rcx,%rax)
addq    $8, %rax
cmpq    %r9, %rax
jne     .L9
```

```
.L9:
①  movq    (%rcx,%rax), %r8
②  movq    (%rdi,%rax), %rsi
③  movq    %r8, (%rdi,%rax)
④  movq    %rsi, (%rcx,%rax)
⑤  addq    $8, %rax
⑥  movq    (%rcx,%rax), %r8
⑦  movq    (%rdi,%rax), %rsi
⑧  cmpq    %r9, %rax
⑨  jne     .L9
⑩  movq    %r8, (%rdi,%rax)
⑪  movq    %rsi, (%rcx,%rax)
⑫  addq    $8, %rax
⑬  cmpq    %r9, %rax
⑭  jne     .L9
```

	IF	ID	ALU/BR/M1	M2	M3	M4/XORL	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(4)	(3)	(2)	(1)			
5	(4)	(3)		(2)	(1)		
6	(4)	(3)			(2)	(1)	
7	(5)	(4)	(3)			(2)	(1)
8	(6)	(5)	(4)	(3)			(2)
9	(7)	(6)	(5)	(4)	(3)		
10	(8)	(7)	(6)	(5)	(4)	(3)	
11	(9)	(8)	(7)	(6)	(5)	(4)	(3)
12	(10)	(9)	(8)	(7)	(6)	(5)	(4)
13	(11)	(10)	(9)	(8)	(7)	(6)	(5)
14	(12)	(11)	(10)	(9)	(8)	(7)	(6)
15	(13)	(12)	(11)	(10)	(9)	(8)	(7)
16	(14)	(13)	(12)	(11)	(10)	(9)	(8)
17		(14)	(13)	(12)	(11)	(10)	(9)
18			(14)	(13)	(12)	(11)	(10)
19				(14)	(13)	(12)	(11)
20					(14)	(13)	(12)
21						(14)	(13)
22							(14)

7 cycles for 7 instructions

CPI = 1

Compiler cannot optimize across branch since it's predicted during runtime!

Recap: What do you need to execution an instruction?

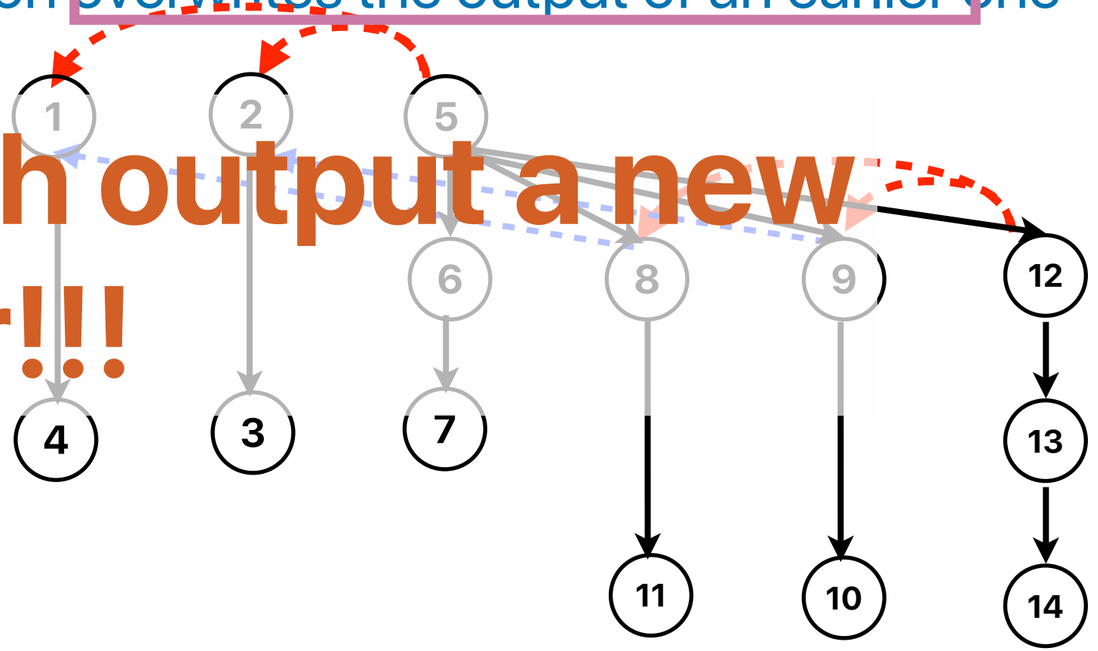
- Whenever the instruction is decoded — put decoded instruction somewhere
- Whenever the inputs are ready — **all data dependencies are resolved**
- Whenever the target functional unit is available

False dependencies

- We are still limited by **false dependencies**
- They are not "true" dependencies because they don't have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 5 and 1, 5 and 2, 12 and 8, 12 and 9
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 8 and 1
    - 9 and 2

We need to give each output a new register!!!

```
①  movq   (%rdi,%rax), %rsi
②  movq   (%rcx,%rax), %r8
③  movq   %r8, (%rdi,%rax)
④  movq   %rsi, (%rcx,%rax)
⑤  addq   $8, %rax
⑥  cmpq   %r9, %rax
⑦  jne    .L9
⑧  movq   (%rdi,%rax), %rsi
⑨  movq   (%rcx,%rax), %r8
⑩  movq   %r8, (%rdi,%rax)
⑪  movq   %rsi, (%rcx,%rax)
⑫  addq   $8, %rax
⑬  cmpq   %r9, %rax
⑭  jne    .L9
```



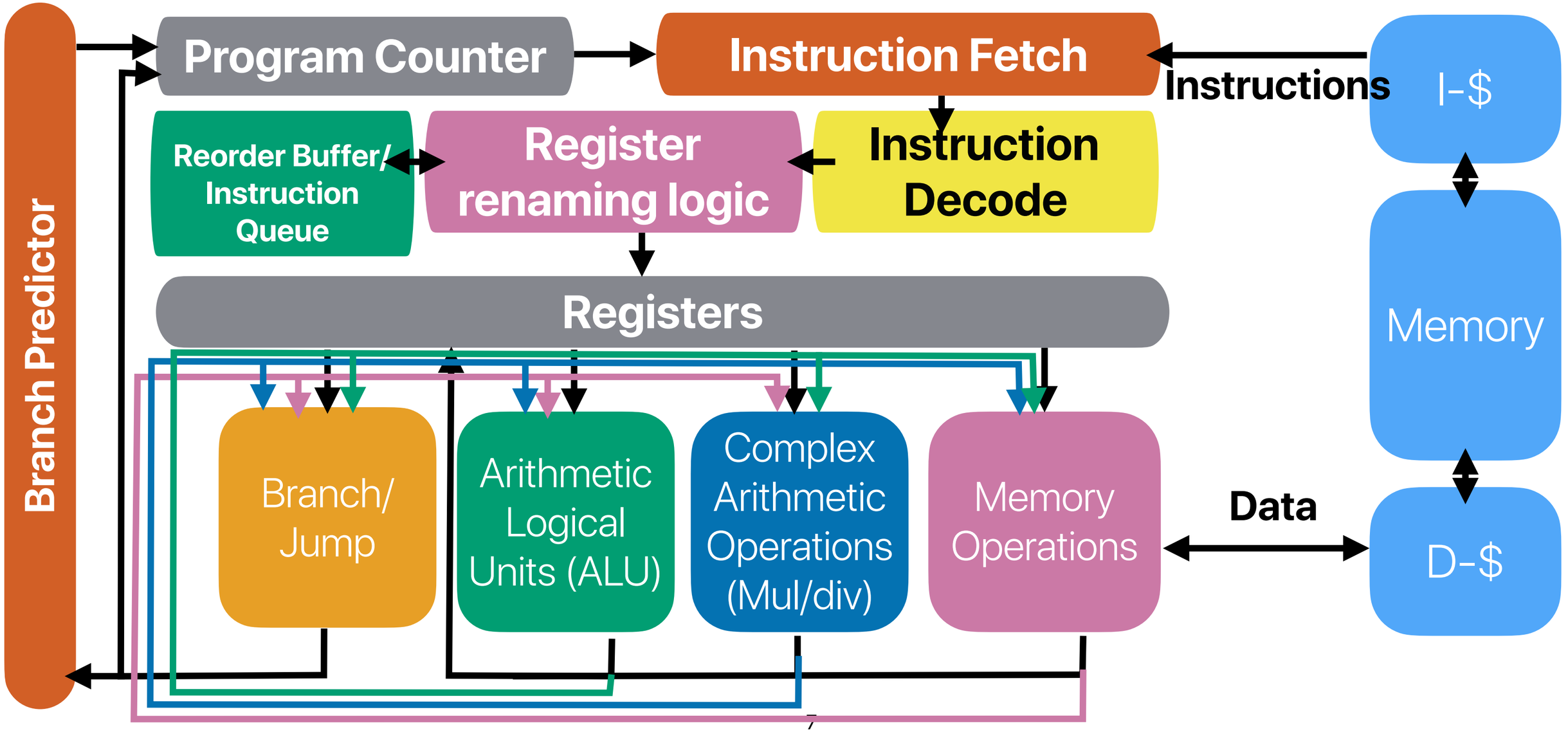
Register renaming + OoO

- Redirecting the output of an instruction instance to a **physical register**
- Redirecting inputs of an instruction instance from **architectural registers** to correct **physical registers**
  - You need a mapping table between architectural and physical registers
  - You may also need reference counters to reclaim physical registers
- OoO: Executing an instruction all operands are ready (the values of depending physical registers are generated)
  - You will need an **issue logic** to **issue** an instruction to the target functional unit

Speculative Execution

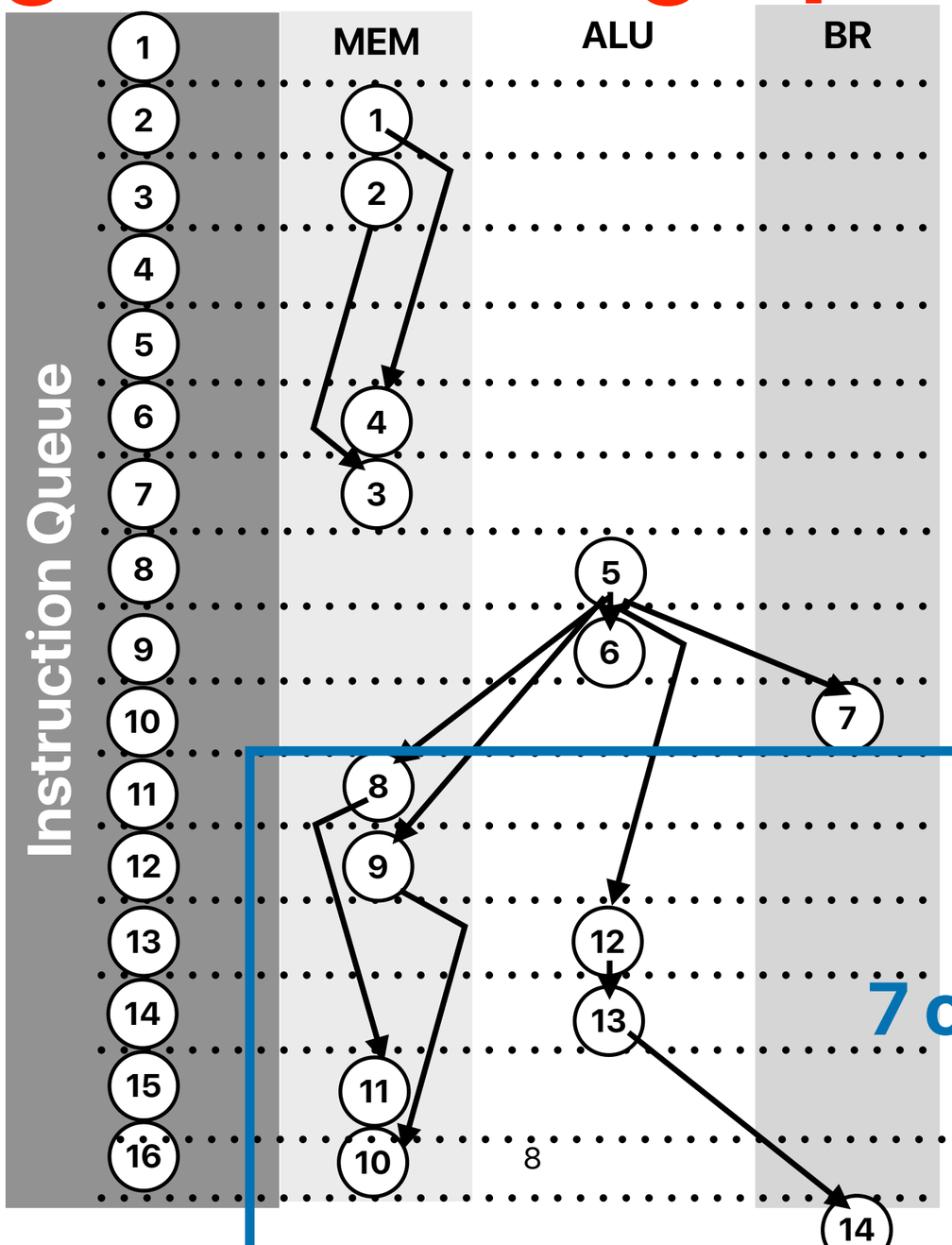
- **Speculative** execution mode: an executing instruction is considered as **speculative** before the processor hasn't determined if the instruction should be executed or not
- Reorder buffer (ROB)
  - The processor allocates an entry for each instruction in a reorder buffer
  - Store results in **reorder buffer and physical registers** when the instruction is still speculative
  - If an earlier instruction failed to commit due to an exception or mis-prediction, the physical registers and all ROB entries after the failed-to-commit instruction are flushed
- Commit/Retire
  - Present the execution result to the running program and in architectural registers when all prior instructions are non-speculative
  - Release the ROB entry

Register renaming + OoO + RoB



Through data flow graph analysis

```
①  movq (%rdi,%rax), %rsi
②  movq (%rcx,%rax), %r8
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi
⑨  movq (%rcx,%rax), %r8
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```



7 cycles every iteration

CPI = $\frac{7}{7} = 1$!

Outline

- How to make CPI < 1
- Code and ILP
- Modern processor design
- Simultaneous multithreading

If CPI==1 the limitation?

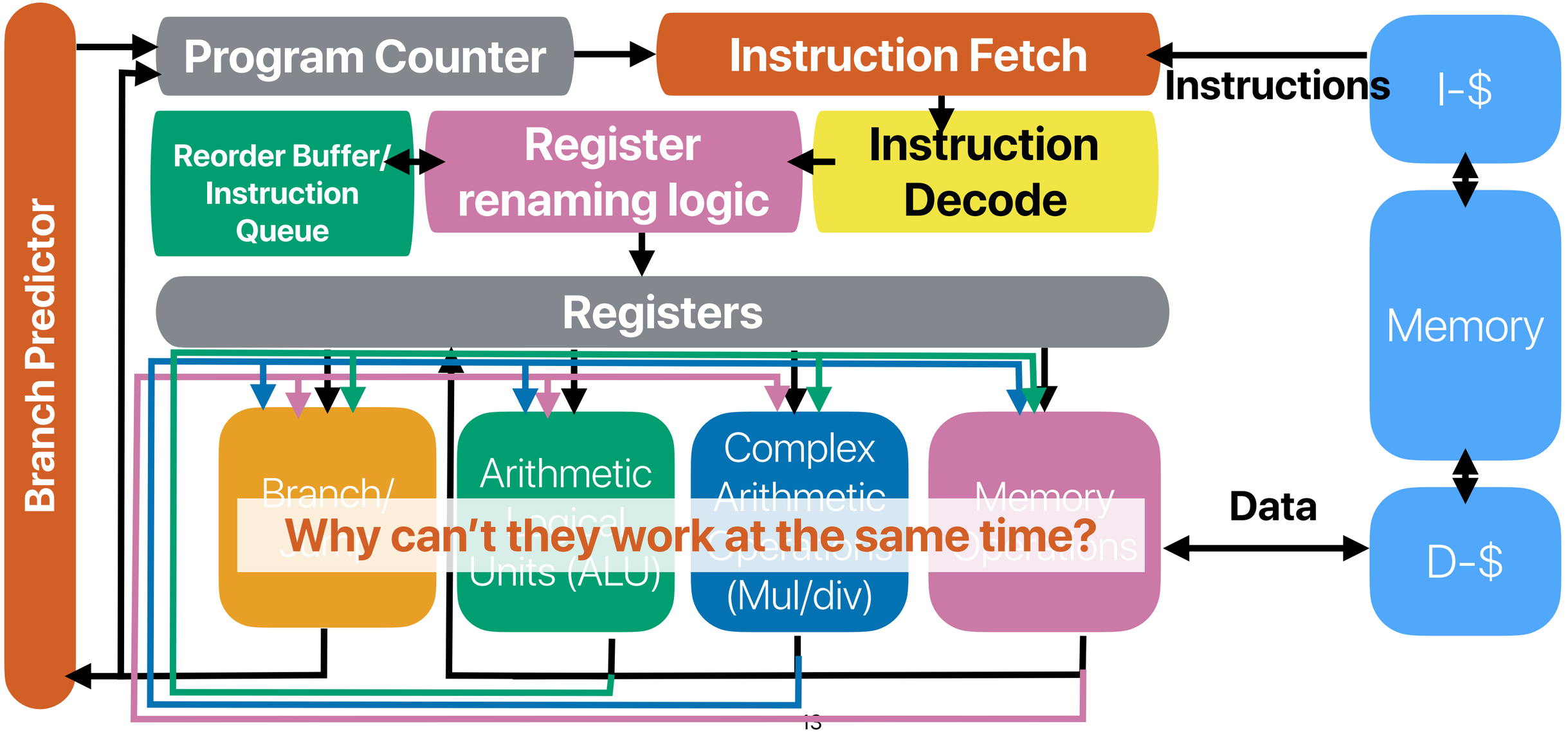
Super Scalar

Superscalar

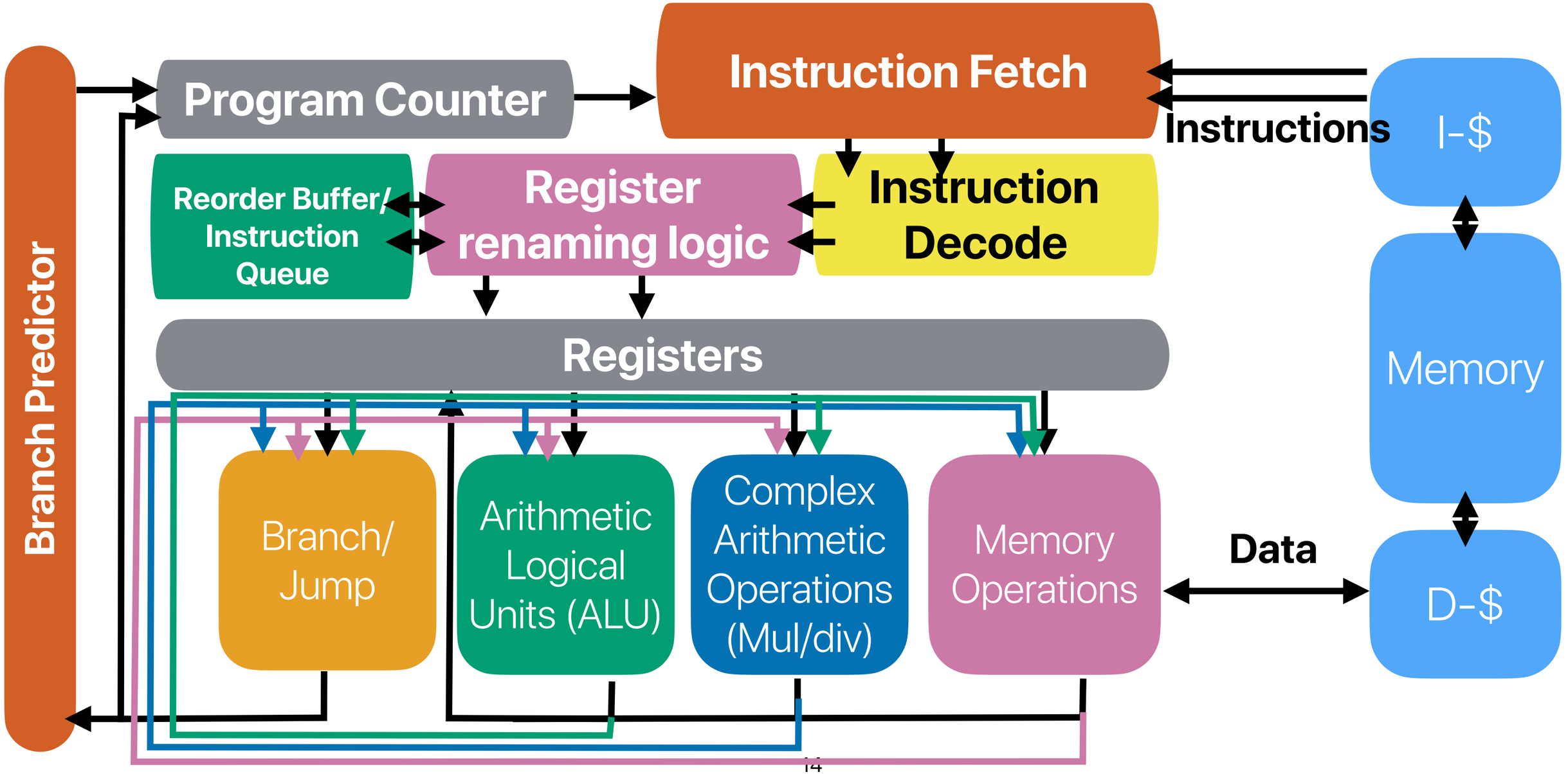
- Since we have many functional units now, we should fetch/decode more instructions each cycle so that we can have more instructions to issue!
- Super-scalar: fetch/decode/issue more than one instruction each cycle
  - **Fetch width:** how many instructions can the processor fetch/decode each cycle
  - **Issue width**: how many instructions can the processor issue each cycle
- The theoretical CPI should now be

$$\frac{1}{min(issue\ \ width, fetch\ \ width, decode\ \ width)}$$

Register renaming + OoO + RoB



Register renaming + SuperScalar



2-issue SS + Register renaming +OoO

Only “2” of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											

2-issue SS + Register renaming +OoO

Only “2” of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											

2-issue SS + Register renaming +OoO

Only “2” of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						

2-issue SS + Register renaming +OoO

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only "2" of them can have a instruction at the same cycle

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			

2-issue SS + Register renaming +OoO

Only “2” of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)

2-issue SS + Register renaming +OoO

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only "2" of them can have a instruction at the same cycle

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	(1)(5)(6)

2-issue SS + Register renaming +OoO

Only “2” of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	(1)(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			(2)(5)(6)(7)

2-issue SS + Register renaming +OoO

Only “2” of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	~~(1)~~(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			~~(2)~~(5)(6)(7)
10	(19)(20)	(17)(18)	(10)(11)(12)(13)(14)(15)(16)	(9)	(3)	(4)	(8)	(13)			(5)(6)(7)(12)

2-issue SS + Register renaming +OoO

Only “2” of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	~~(1)~~(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			~~(2)~~(5)(6)(7)
10	(19)(20)	(17)(18)	(10)(11)(12)(13)(14)(15)(16)	(9)	(3)	(4)	(8)	(13)			(5)(6)(7)(12)
11		(19)(20)	(10)(12)(13)(15)(16)(17)(18)	(11)	(9)	(3)	(4)			(14)	(5)(6)(7)(8)(12)(13)

2-issue SS + Register renaming +OoO

Only "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	~~(1)~~(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			~~(2)~~(5)(6)(7)
10	(19)(20)	(17)(18)	(10)(11)(12)(13)(14)(15)(16)	(9)	(3)	(4)	(8)	(13)			(5)(6)(7)(12)
11		(19)(20)	(10)(12)(13)(15)(16)(17)(18)	(11)	(9)	(3)	(4)			(14)	(5)(6)(7)(8)(12)(13)
12			(12)(13)(15)(16)(17)(18)(19)(20)	(10)	(11)	(9)	(3)				(4)(5)(6)(7)(8)(12)(13)(14)

2-issue SS + Register renaming +OoO

Only "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	~~(1)~~(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			~~(2)~~(5)(6)(7)
10	(19)(20)	(17)(18)	(10)(11)(12)(13)(14)(15)(16)	(9)	(3)	(4)	(8)	(13)			(5)(6)(7)(12)
11		(19)(20)	(10)(12)(13)(15)(16)(17)(18)	(11)	(9)	(3)	(4)			(14)	(5)(6)(7)(8)(12)(13)
12			(12)(13)(15)(16)(17)(18)(19)(20)	(10)	(11)	(9)	(3)				(4)(5)(6)(7)(8)(12)(13)(14)
13			(12)(13)(16)(17)(18)(20)	(15)	(10)	(11)	(9)	(19)			~~(3)(4)(5)(6)(7)(8)~~(12)(13)(14)

2-issue SS + Register renaming +OoO

Only "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	~~(1)~~(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			~~(2)~~(5)(6)(7)
10	(19)(20)	(17)(18)	(10)(11)(12)(13)(14)(15)(16)	(9)	(3)	(4)	(8)	(13)			(5)(6)(7)(12)
11		(19)(20)	(10)(12)(13)(15)(16)(17)(18)	(11)	(9)	(3)	(4)			(14)	(5)(6)(7)(8)(12)(13)
12			(12)(13)(15)(16)(17)(18)(19)(20)	(10)	(11)	(9)	(3)				(4)(5)(6)(7)(8)(12)(13)(14)
13			(12)(13)(16)(17)(18)(20)	(15)	(10)	(11)	(9)	(19)			~~(3)(4)(5)(6)(7)(8)~~(12)(13)(14)
14			(12)(13)(17)(18)	(16)	(15)	(10)	(11)	(20)			(9)(12)(13)(14)(19)

2-issue SS + Register renaming +OoO

Only "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	~~(1)~~(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			~~(2)~~(5)(6)(7)
10	(19)(20)	(17)(18)	(10)(11)(12)(13)(14)(15)(16)	(9)	(3)	(4)	(8)	(13)			(5)(6)(7)(12)
11		(19)(20)	(10)(12)(13)(15)(16)(17)(18)	(11)	(9)	(3)	(4)			(14)	(5)(6)(7)(8)(12)(13)
12			(12)(13)(15)(16)(17)(18)(19)(20)	(10)	(11)	(9)	(3)				(4)(5)(6)(7)(8)(12)(13)(14)
13			(12)(13)(16)(17)(18)(20)	(15)	(10)	(11)	(9)	(19)			~~(3)(4)(5)(6)(7)(8)~~(12)(13)(14)
14			(12)(13)(17)(18)	(16)	(15)	(10)	(11)	(20)			~~(9)~~(12)(13)(14)(19)
15					(16)	(15)	(10)			(21)	(11)(12)(13)(14)(19)(20)

2-issue SS + Register renaming +OoO

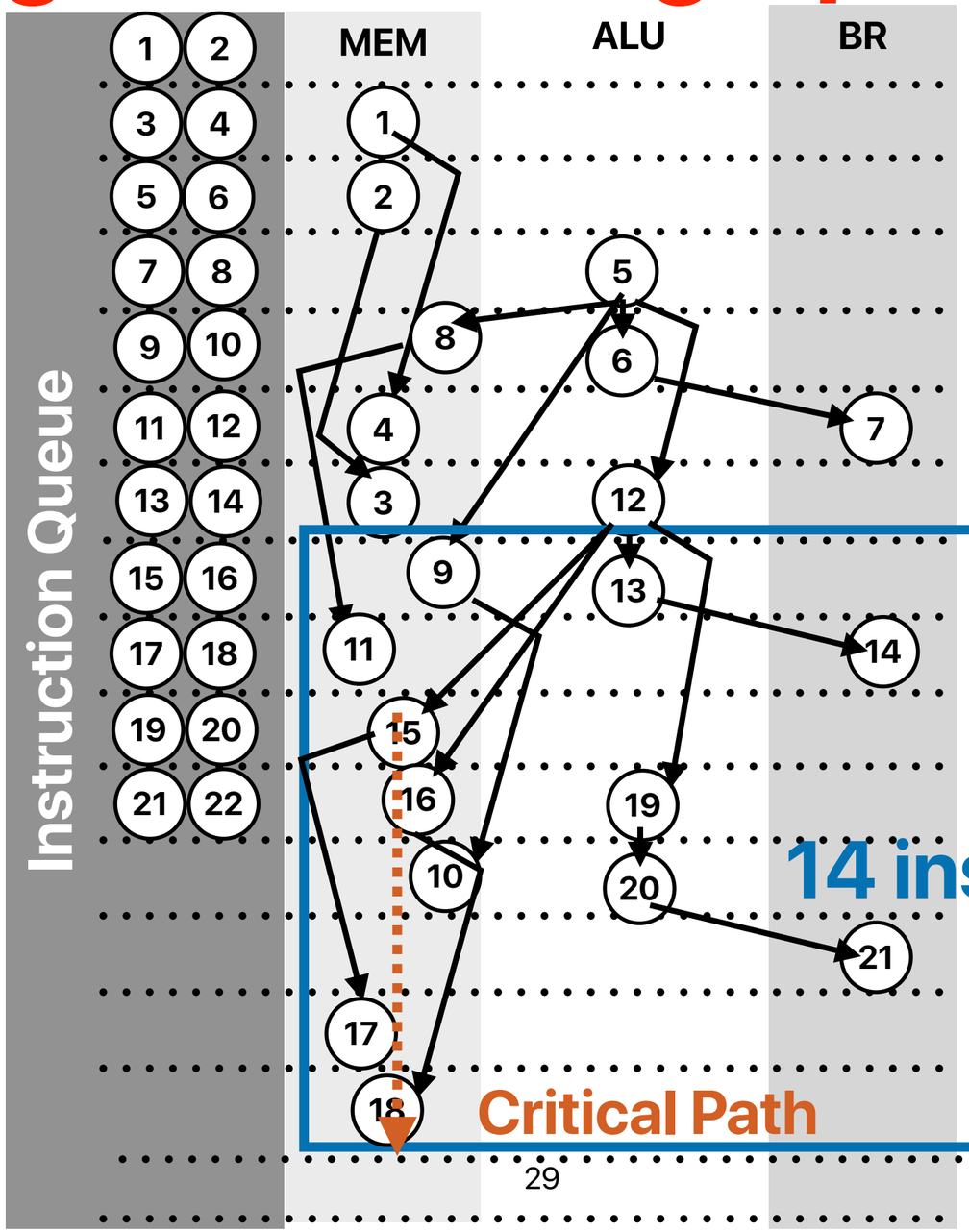
Only "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)										
2	(3)(4)	(1)(2)									
3	(5)(6)	(3)(4)	(1)(2)								
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)							
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)						
6	(11)(12)	(9)(10)	(3)(4)(6)(7)(8)		(2)	(1)		(5)			
7	(13)(14)	(11)(12)	(3)(4)(7)(9)(10)	(8)		(2)	(1)	(6)			(5)
8	(15)(16)	(13)(14)	(3)(9)(10)(11)(12)	(4)	(8)		(2)			(7)	~~(1)~~(5)(6)
9	(17)(18)	(15)(16)	(9)(10)(11)(12)(13)(14)	(3)	(4)	(8)		(12)			~~(2)~~(5)(6)(7)
10	(19)(20)	(17)(18)	(10)(11)(12)(13)(14)(15)(16)	(9)	(3)	(4)	(8)	(13)			(5)(6)(7)(12)
11		(19)(20)	(10)(12)(13)(15)(16)(17)(18)	(11)	(9)	(3)	(4)			(14)	(5)(6)(7)(8)(12)(13)
12			(12)(13)(15)(16)(17)(18)(19)(20)	(10)	(11)	(9)	(3)				(4)(5)(6)(7)(8)(12)(13)(14)
13			(12)(13)(16)(17)(18)(20)	(15)	(10)	(11)	(9)	(19)			~~(3)(4)(5)(6)(7)(8)~~(12)(13)(14)
14			(12)(13)(17)(18)	(16)	(15)	(10)	(11)	(20)			~~(9)~~(12)(13)(14)(19)
15					(16)	(15)	(10)			(21)	(11)(12)(13)(14)(19)(20)
16						(16)	(15)				~~(10)(11)(12)(13)(14)~~(19)(20)(21)

Through data flow graph analysis

```
①  movq (%rdi,%rax), %rsi
②  movq (%rcx,%rax), %r8
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi
⑨  movq (%rcx,%rax), %r8
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```



14 instructions in 8 cycles

CPI = $\frac{8}{14} = 0.57$**!**

What about “linked list”

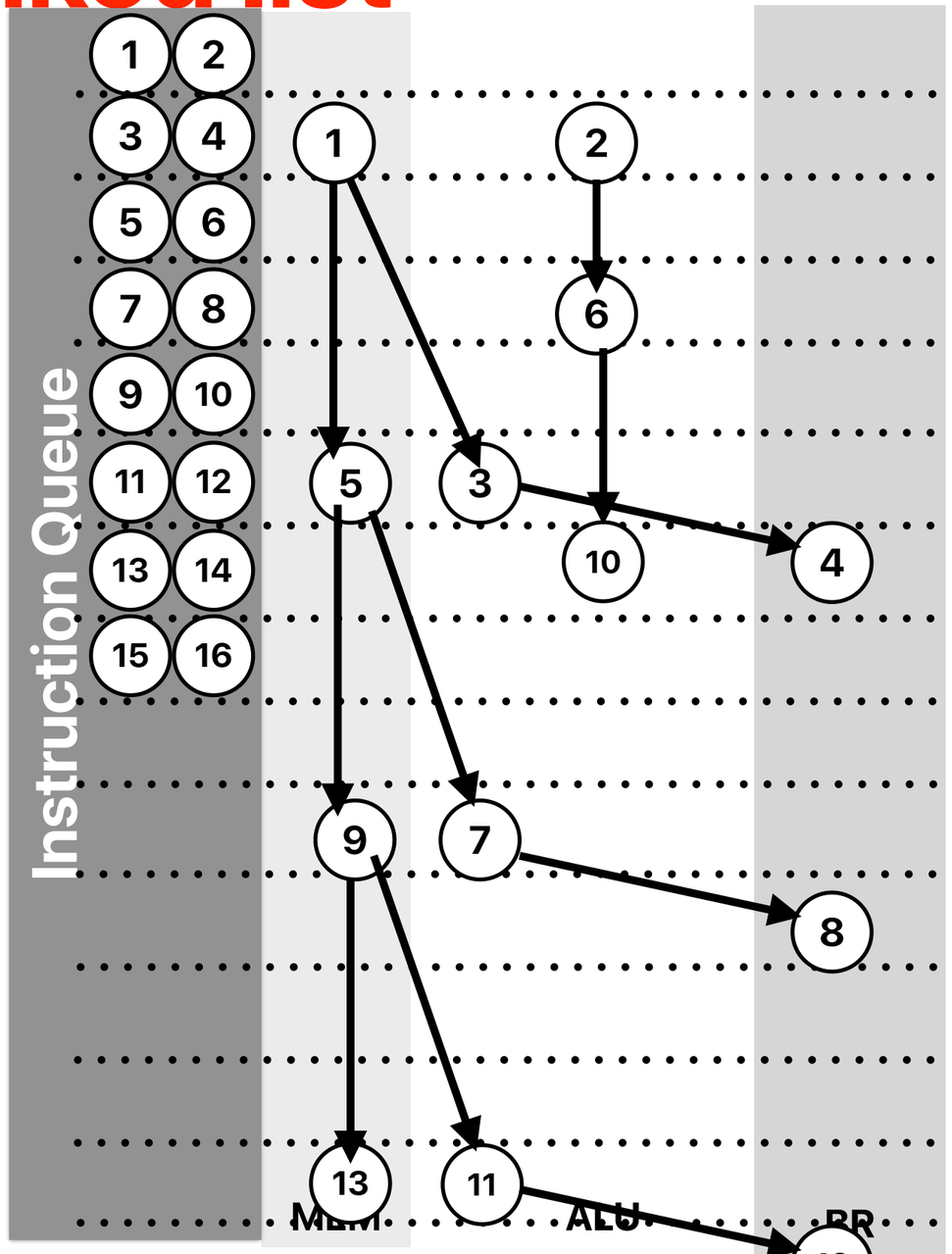
- Assume the current PC is already at instruction (1) and this linked list has only three nodes. This processor can fetch and issue 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.
  Which of the following C state of the code snippet determines the performance?

```
A.do {
B.     number_of_nodes++;
C.     current = current->next;
D.} while ( current != NULL );
```

What about “linked list”

Dynamic instructions

```
①  .L3:    movq    8(%rdi), %rdi
②          addl    $1, %eax
③          testq   %rdi, %rdi
④          jne     .L3
⑤  .L3:    movq    8(%rdi), %rdi
⑥          addl    $1, %eax
⑦          testq   %rdi, %rdi
⑧          jne     .L3
⑨  .L3:    movq    8(%rdi), %rdi
⑩          addl    $1, %eax
⑪          testq   %rdi, %rdi
⑫          jne     .L3
⑬  .L3:    movq    8(%rdi), %rdi
⑭          addl    $1, %eax
⑮          testq   %rdi, %rdi
⑯          jne     .L3
```

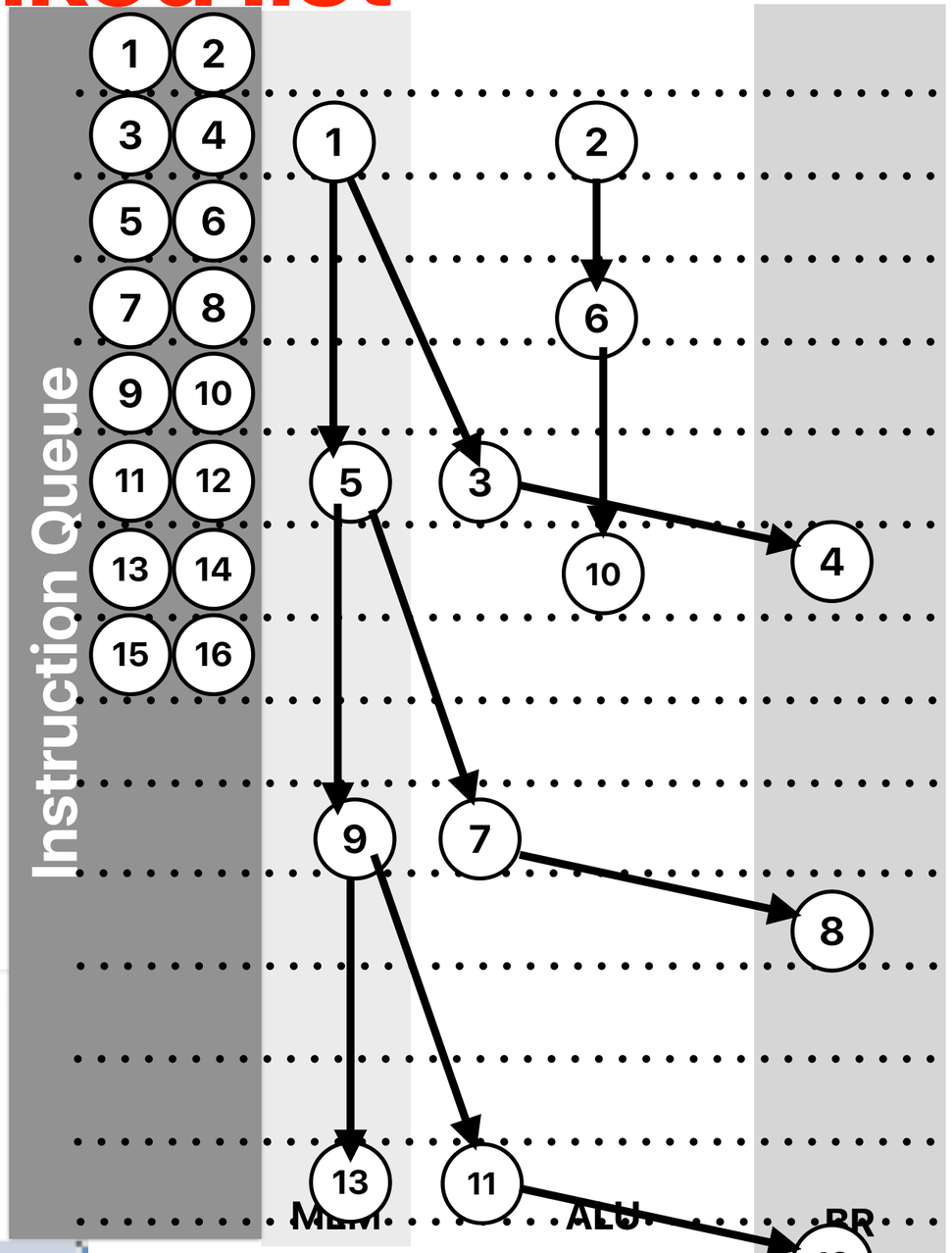


What about “linked list”

- For the following C code and it’s translation in x86, **what’s average CPI?** Assume the current PC is already at instruction (1) and this linked list has thousands of nodes. This processor can fetch and issue **2** instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL )
```

A. 0.5
B. 0.8
C. 1.0
D. 1.2
E. 1.5



What about “linked list”

Performance determined by the critical path

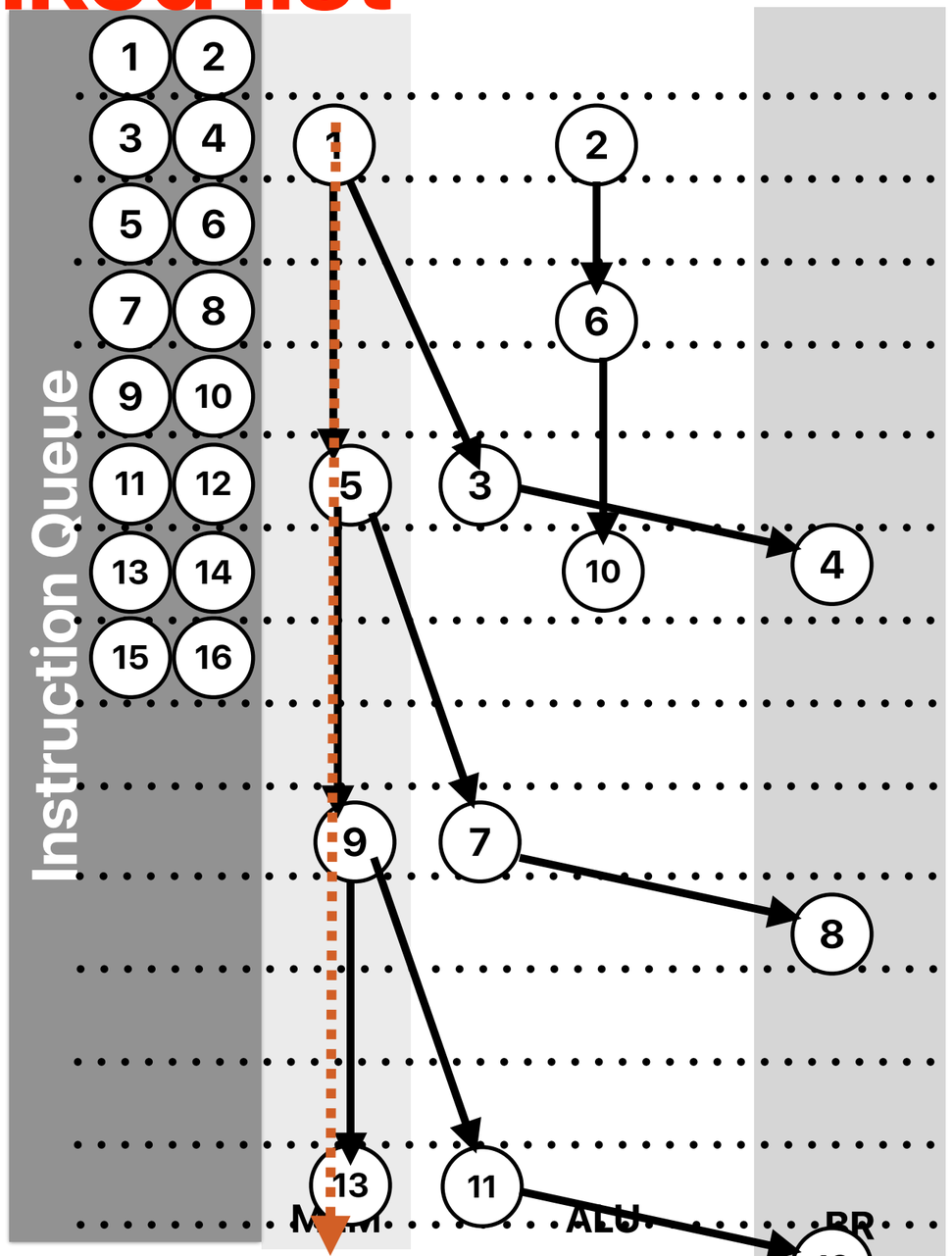
4 cycles each iteration

4 instructions per iteration

$$CPI = \frac{4}{4} = 1$$

```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

```
① .L3:    movq    8(%rdi), %rdi
②         addl    $1, %eax
③         testq   %rdi, %rdi
④         jne     .L3
```



What about "arrays?"

Performance determined by the critical path

3 cycles each iteration

5 instructions per iteration

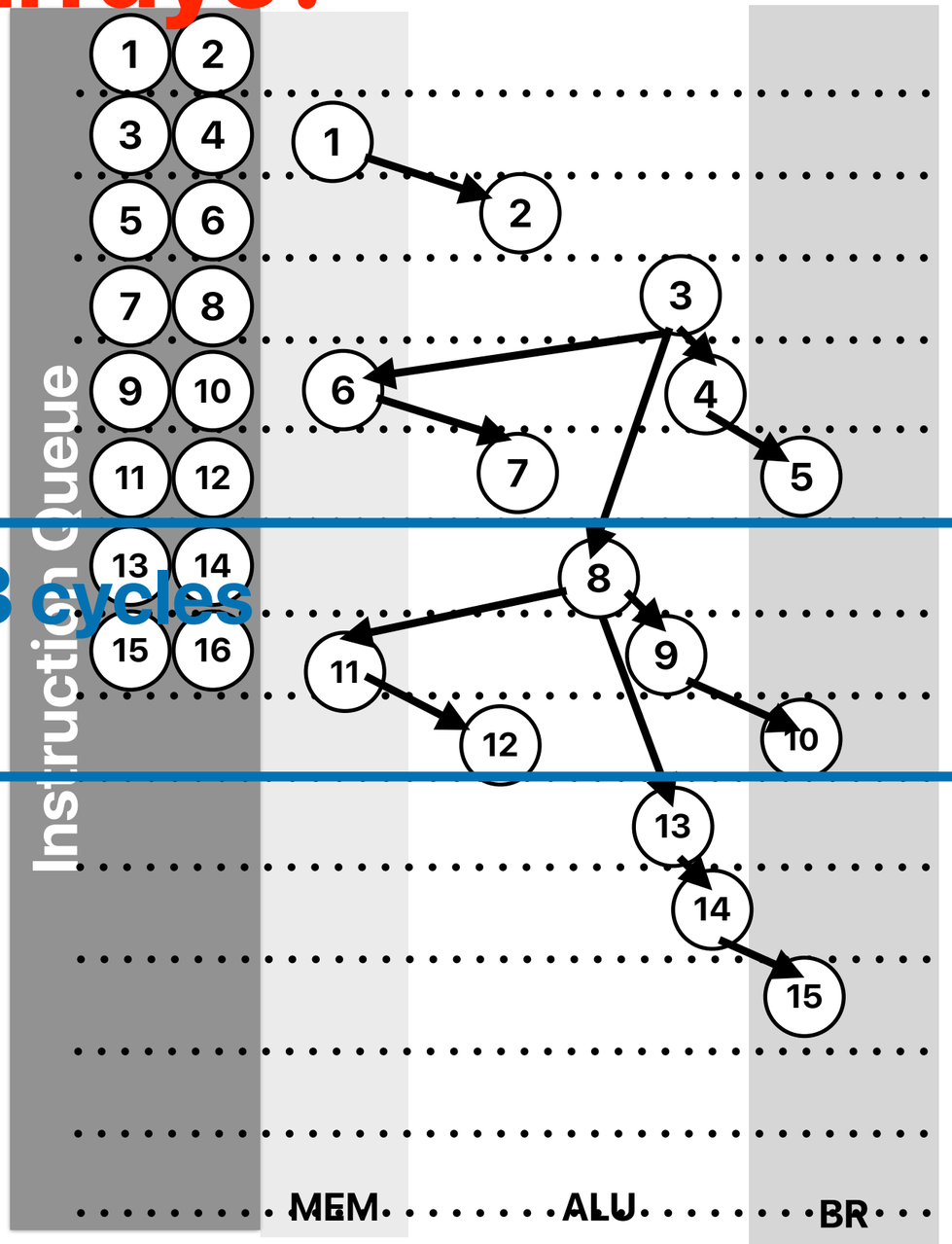
$$CPI = \frac{3}{5} = 0.6$$

```
for(i=0;i<size;i++) {
    if(node[i].next)
        number_of_nodes++;
}
```

```
① .L9: cmpq $1, 8(%rax)
②      sbbl $-1, %edx
③      addq $16, %rax
④      cmpq %rdi, %rax
⑤      jne  .L9
```

5 instructions in 3 cycles

CPI = $\frac{3}{5} = 0.6$**!**

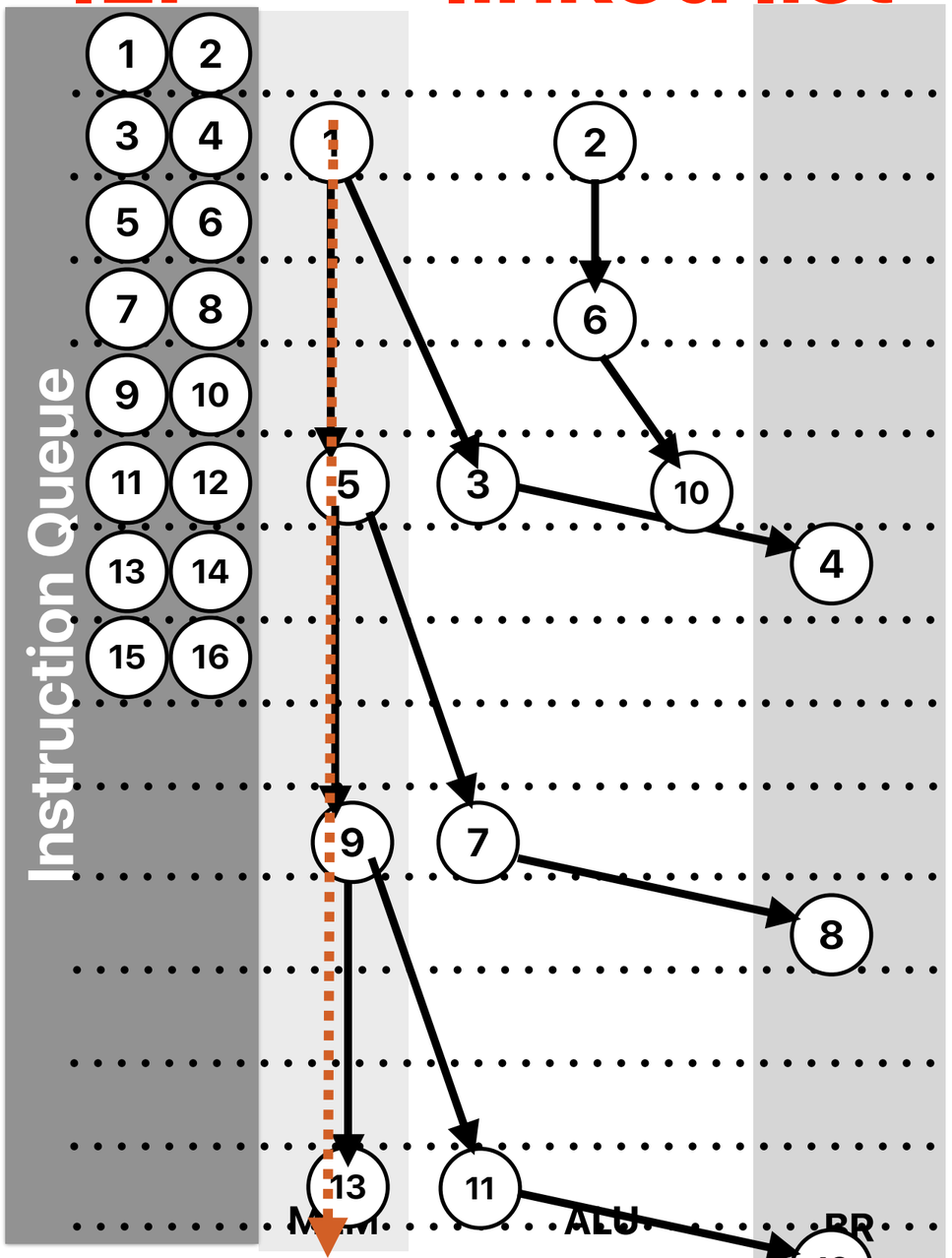


What if we have “unlimited” ILP — “linked list”

Doesn’t help that much!
— It’s important that the programmer should write code that can exploit “ILP”

```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

```
① .L3:    movq    8(%rdi), %rdi
②         addl    $1, %eax
③         testq   %rdi, %rdi
④         jne     .L3
```



The pipelines of Modern Processors

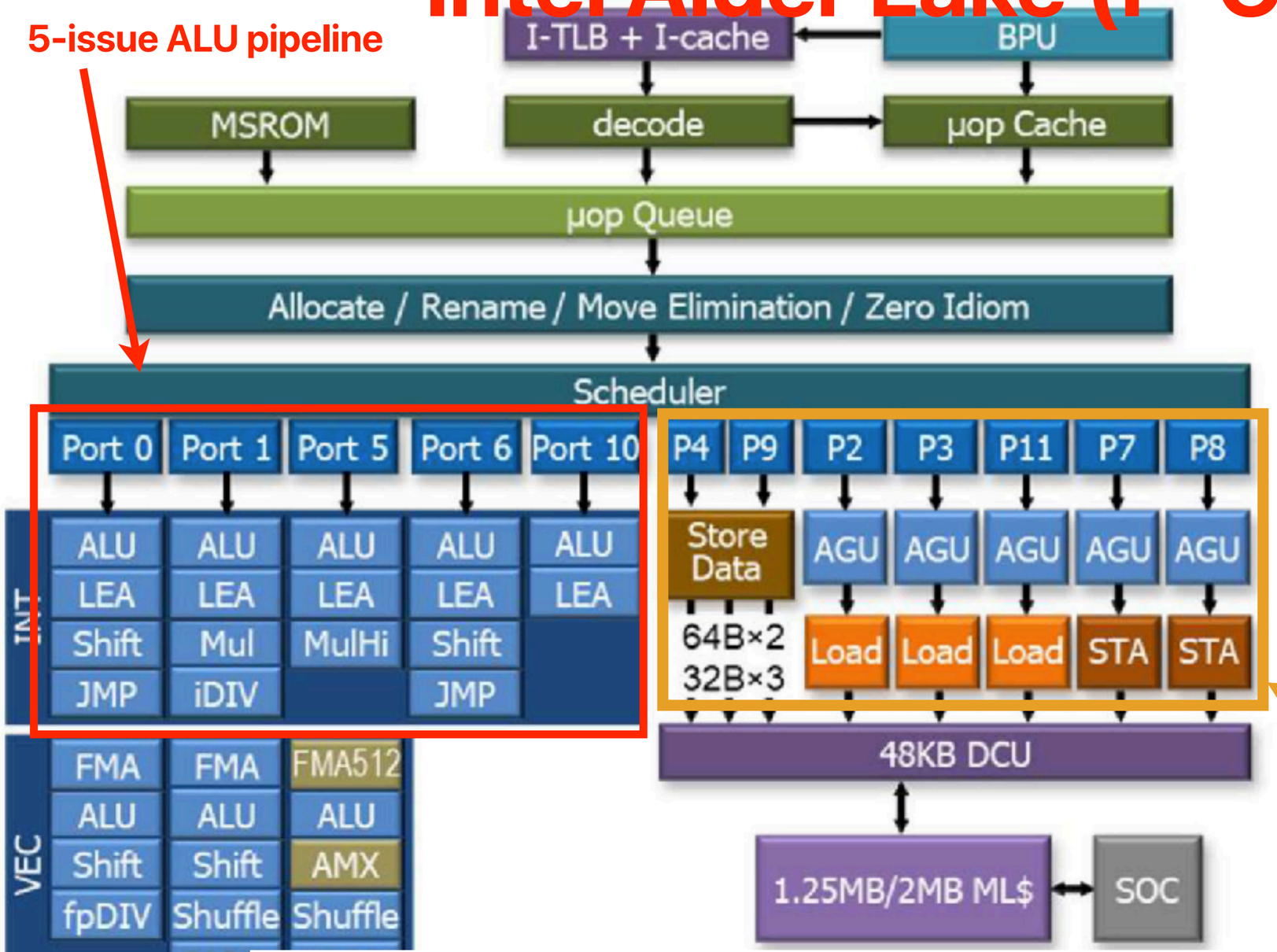
Intel Alder Lake (P-Core)

$$MinCPI = \frac{1}{12}$$

$$MinINTInst.CPI = \frac{1}{5}$$

$$MinMEMInst.CPI = \frac{1}{7}$$

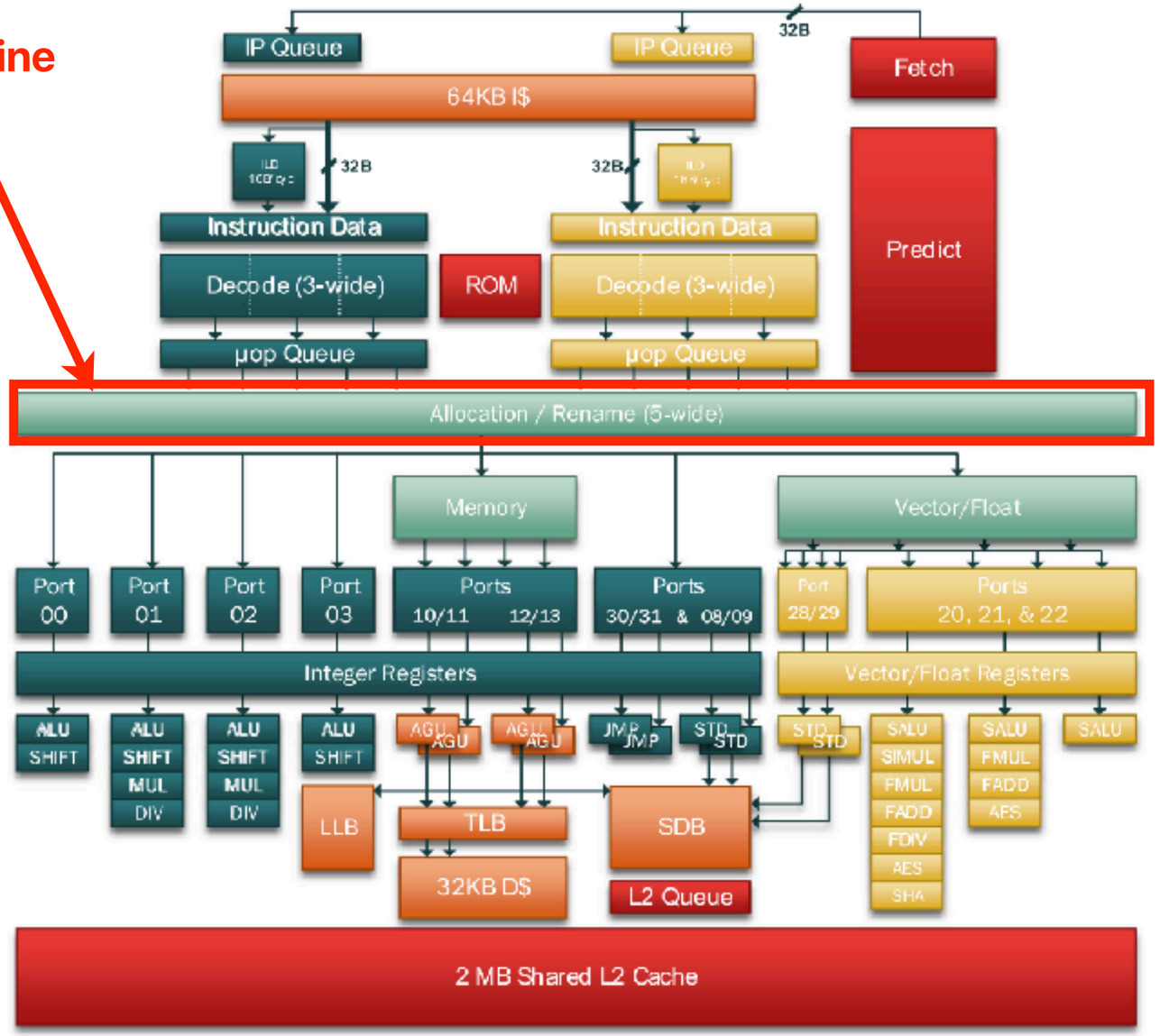
$$MinBRInst.CPI = \frac{1}{2}$$



https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Intel Alder Lake (E-Core)

5-issue pipeline



AMD Zen 3 (RyZen 5000 Series)

3-issue memory pipeline

4-issue integer pipeline + 1 additional branch

$$MinCPI = \frac{1}{8}$$

$$MinINTInst\,.\,CPI = \frac{1}{4}$$

$$MinMEMInst\,.\,CPI = \frac{1}{3}$$

$$MinBRInst\,.\,CPI = \frac{1}{2}$$

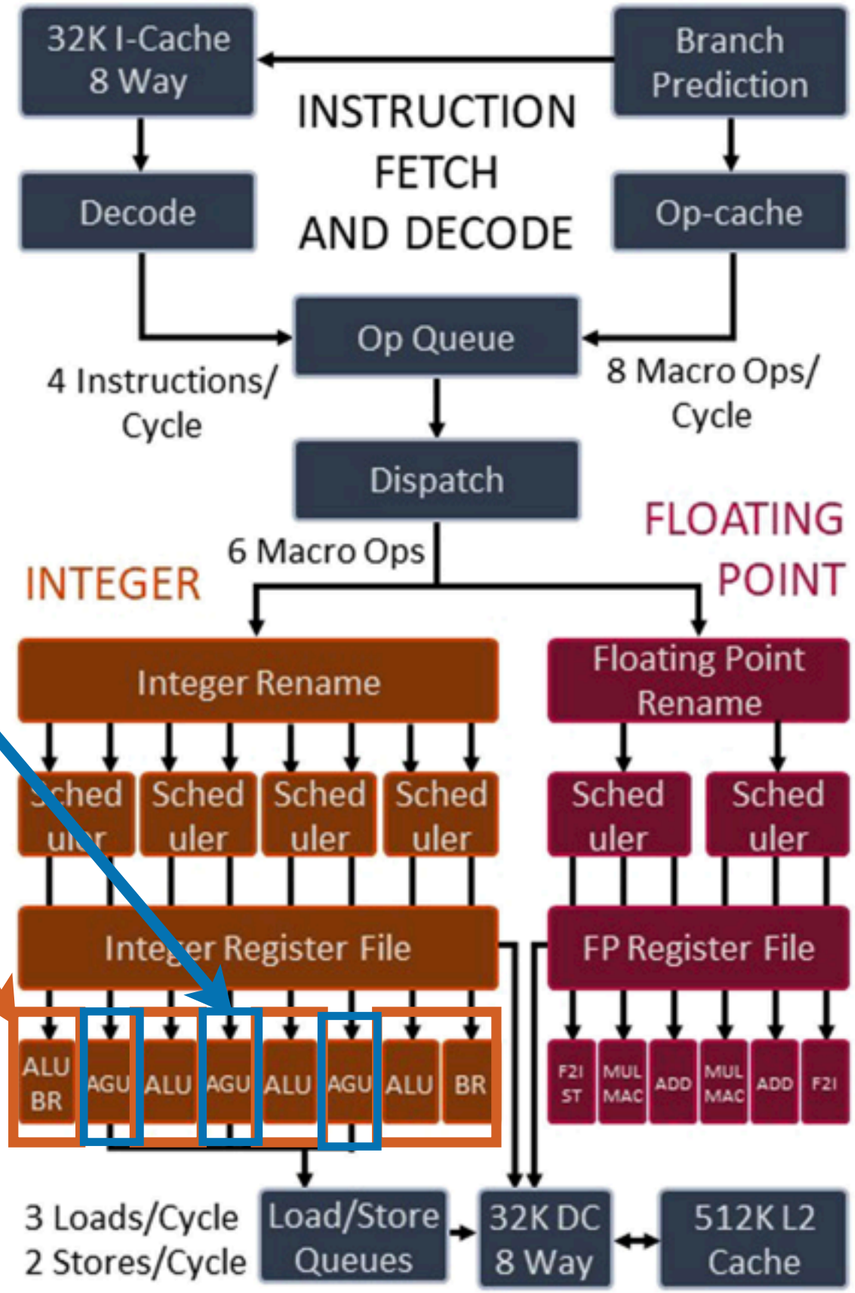


FIGURE 1. "Zen 3" block diagram.

Takeaways: Tips of programming on modern processors

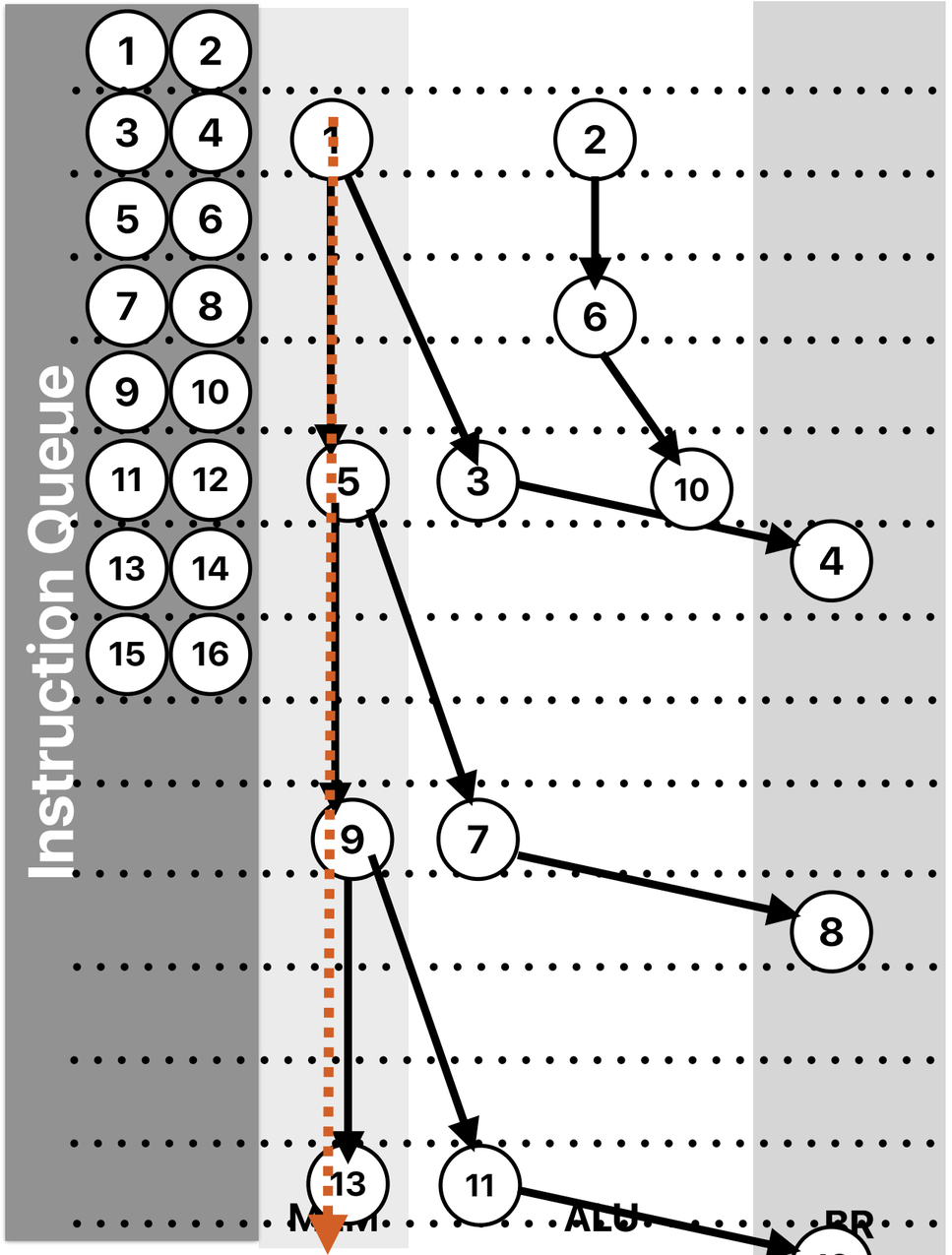
- Minimize the critical path operations
  - Don’t forget about optimizing cache/memory locality first!
    - Memory latencies are still way longer than any arithmetic instruction
    - Can we use arrays/hash tables instead of lists?
  - Branch can be expensive as pipeline get deeper
    - Sorting
    - Loop unrolling
  - Still need to carefully avoid long latency operations (e.g., mod)
- Since processors have multiple functional units — code must be able to exploit instruction-level parallelism
  - Hide as many instructions as possible under the “critical path”
  - Try to use as many different functional units simultaneously as possible
- Modern processors also have accelerated instructions
- Compiler can do fairly go optimizations, but with limitations

What if we have “unlimited” fetch/issue width — “linked list”

Doesn’t help that much!
— It’s important that the programmer should write code that can exploit “ILP”
— But — there’re always cases we cannot do further in ILP

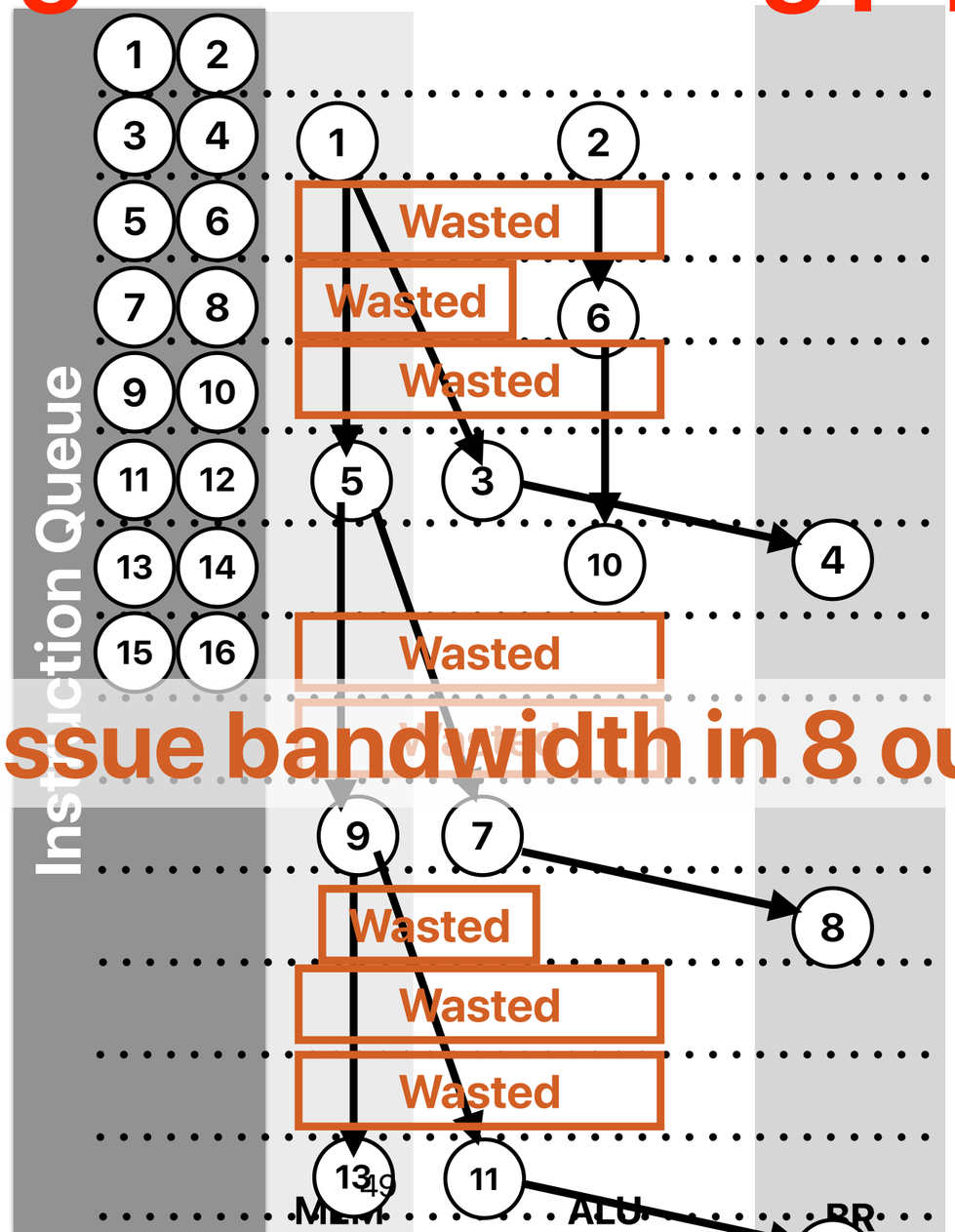
```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

```
① .L3:    movq    8(%rdi), %rdi
②         addl    $1, %eax
③         testq   %rdi, %rdi
④         jne     .L3
```



2-issue register renaming pipeline

```
① movq    8(%rdi), %rdi
② addl    $1, %eax
③ testq   %rdi, %rdi
④ jne     .L3
⑤ movq    8(%rdi), %rdi
⑥ addl    $1, %eax
⑦ testq   %rdi, %rdi
⑧ jne     .L3
⑨ movq    8(%rdi), %rdi
⑩ addl    $1, %eax
⑪ testq   %rdi, %rdi
⑫ jne     .L3
```

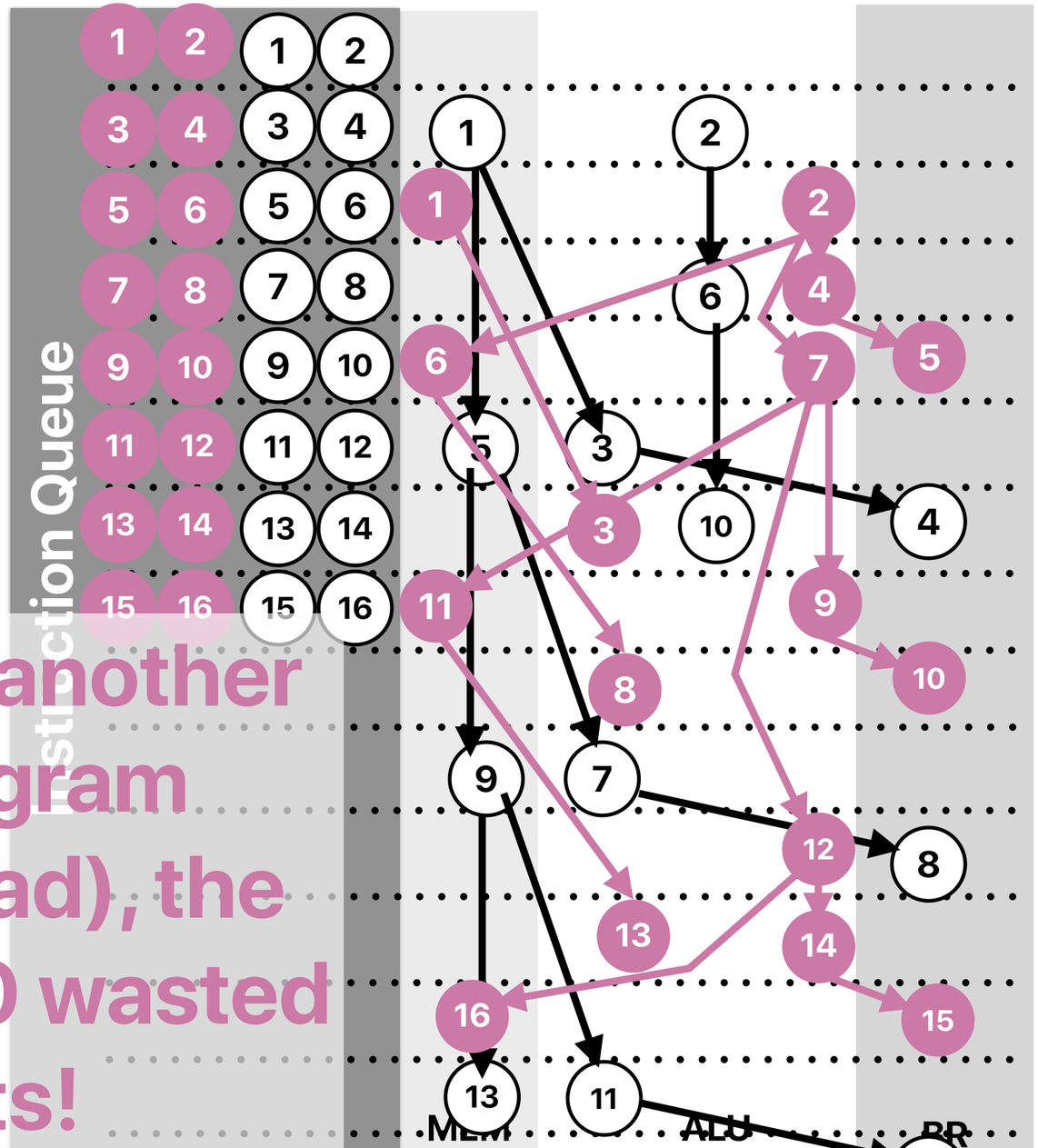


We're wasting the issue bandwidth in 8 out of 12 cycles

Simultaneous multithreading

Concept: Simultaneous Multithreading (SMT)

```
①  movq    8(%rdi), %rdi
②  addl    $1, %eax
③  testq   %rdi, %rdi
④  jne     .L3
⑤  movq    8(%rdi), %rdi
⑥  addl    $1, %eax
⑦  testq   %rdi, %rdi
⑧  jne     .L3
⑨  movq    8(%rdi), %rdi
⑩  addl    $1, %eax
⑪  testq   %rdi, %rdi
⑫  jne     .L3
```

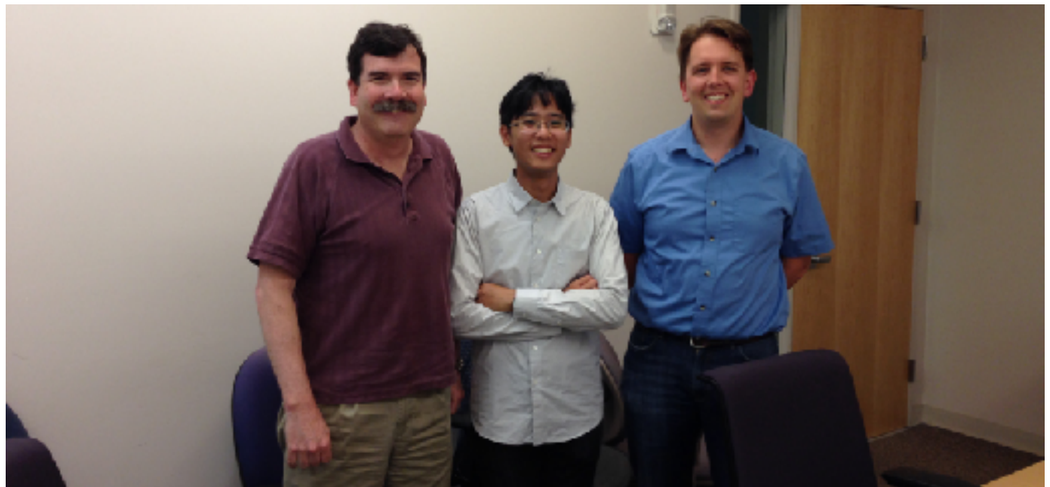


By scheduling another running program instance (thread), the processor has 0 wasted issue slots!

```
①  movl (%rdi), %ecx
②  addq $4, %rdi
③  addl %ecx, %eax
④  cmpq %rdx, %rdi
⑤  jne  .L3
⑥  movl (%rdi), %ecx
⑦  addq $4, %rdi
⑧  addl %ecx, %eax
⑨  cmpq %rdx, %rdi
⑩  jne  .L3
⑪  movl (%rdi), %ecx
⑫  addq $4, %rdi
⑬  addl %ecx, %eax
⑭  cmpq %rdx, %rdi
⑮  jne  .L3
⑯  movl (%rdi), %ecx
```

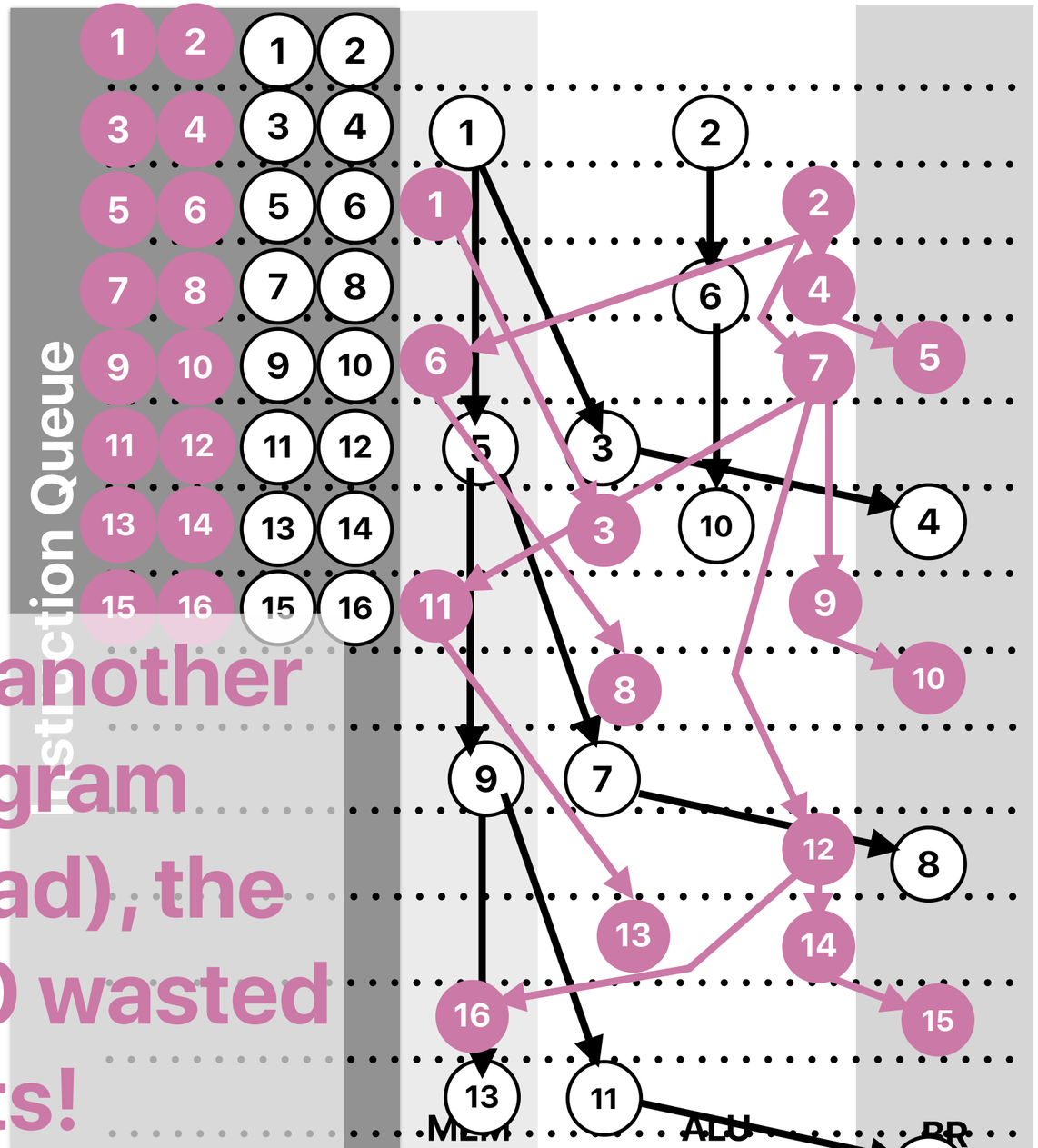
Simultaneous multithreading

- The processor can schedule instructions from different threads/processes/ programs
- Fetch instructions from different threads/processes to fill the not utilized part of pipeline
  - Exploit “thread level parallelism” (TLP) to solve the problem of insufficient ILP in a single thread
  - You need to create an illusion of multiple processors for OSs
- Invented by Dean Tullsen (Now a professor at **UCSD CSE**)



Concept: Simultaneous Multithreading (SMT)

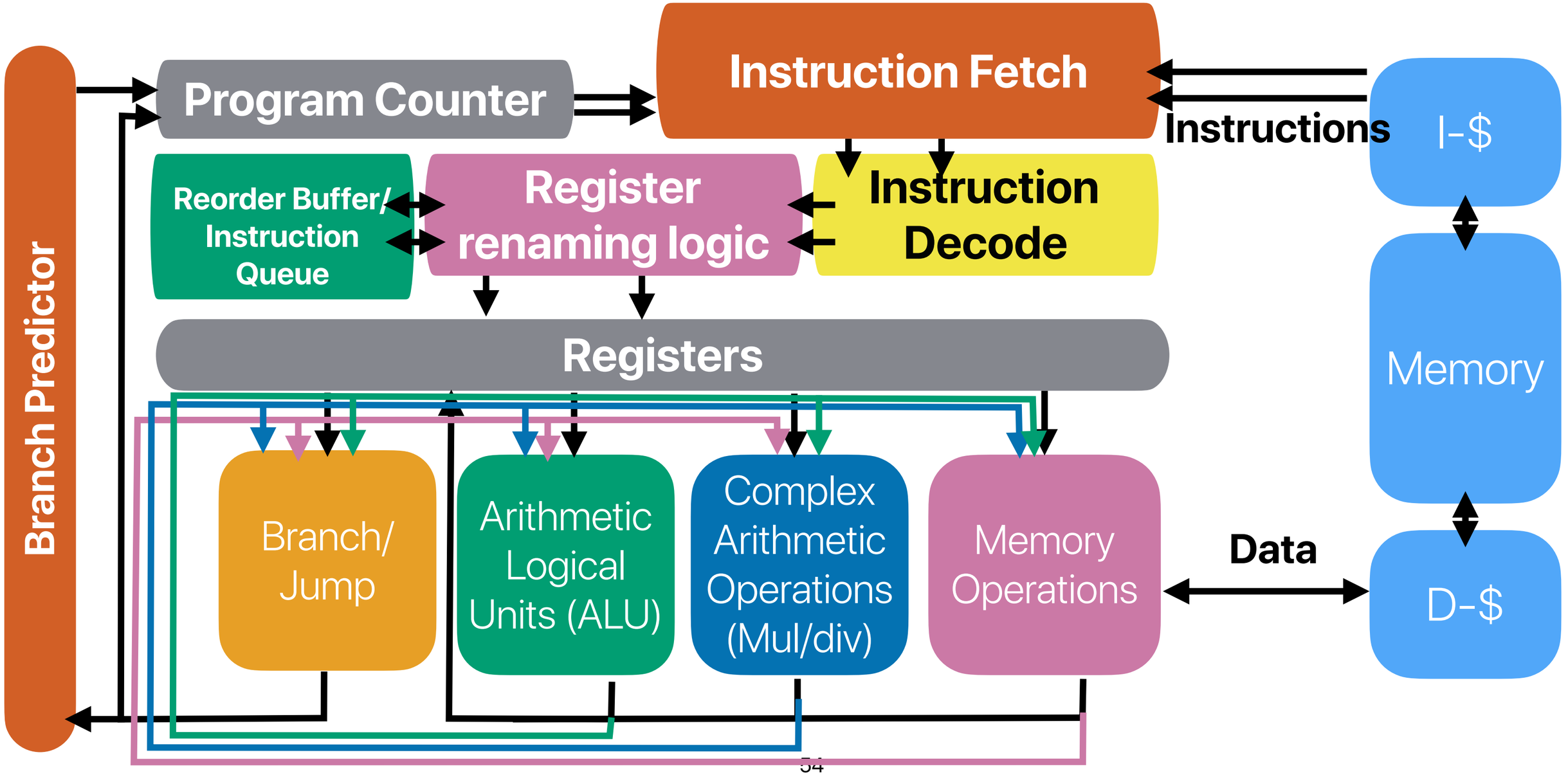
```
①  movq    8(%rdi), %rdi
②  addl    $1, %eax
③  testq   %rdi, %rdi
④  jne     .L3
⑤  movq    8(%rdi), %rdi
⑥  addl    $1, %eax
⑦  testq   %rdi, %rdi
⑧  jne     .L3
⑨  movq    8(%rdi), %rdi
⑩  addl    $1, %eax
⑪  testq   %rdi, %rdi
⑫  jne     .L3
```



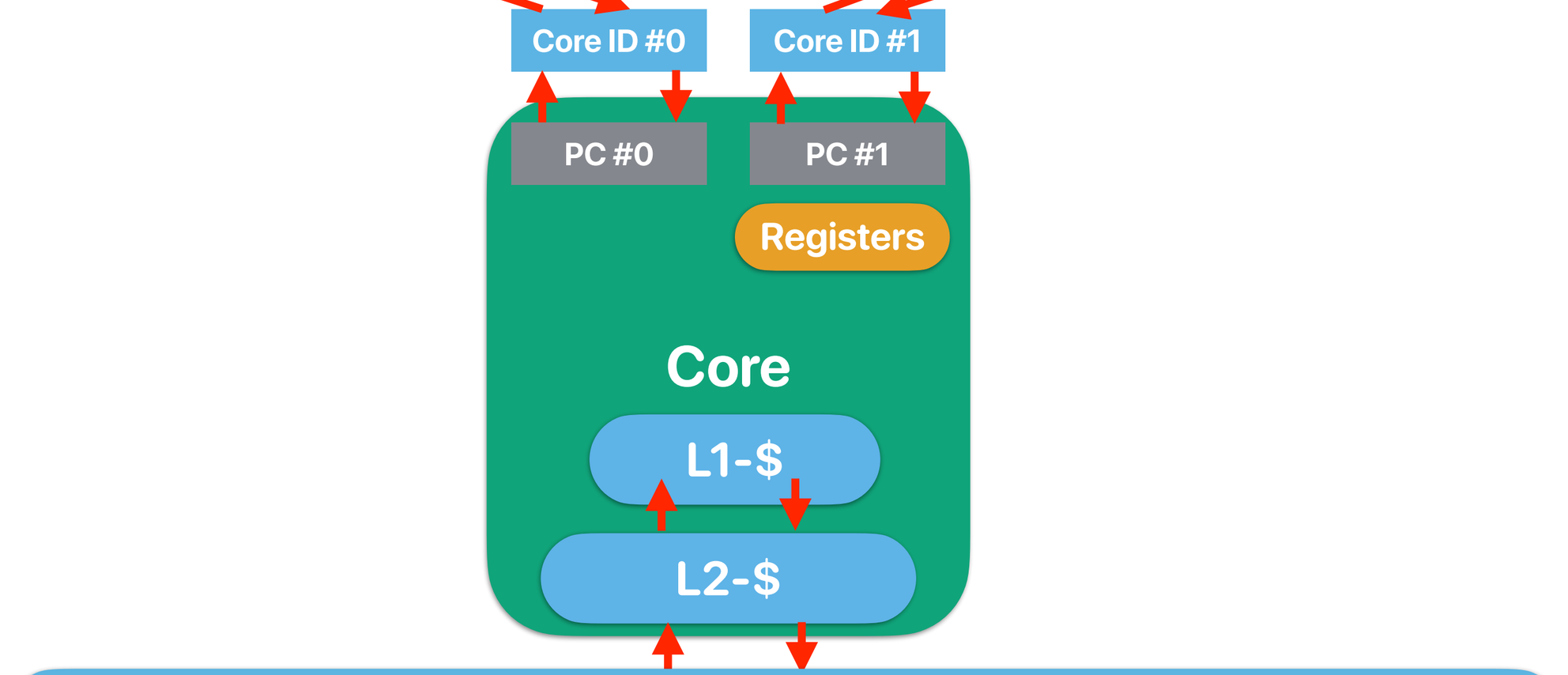
By scheduling another running program instance (thread), the processor has 0 wasted issue slots!

```
①  movl (%rdi), %ecx
②  addq $4, %rdi
③  addl %ecx, %eax
④  cmpq %rdx, %rdi
⑤  jne  .L3
⑥  movl (%rdi), %ecx
⑦  addq $4, %rdi
⑧  addl %ecx, %eax
⑨  cmpq %rdx, %rdi
⑩  jne  .L3
⑪  movl (%rdi), %ecx
⑫  addq $4, %rdi
⑬  addl %ecx, %eax
⑭  cmpq %rdx, %rdi
⑮  jne  .L3
⑯  movl (%rdi), %ecx
```

Recap: Register renaming



SMT from the user/OS' perspective



https://www.pollev.com/hungweitseng close in 1:30

Architectural support for simultaneous multithreading

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter
  - ② Register mapping tables
  - ③ Physical registers
  - ④ ALUs
  - ⑤ Data cache
  - ⑥ Reorder buffer/Instruction Queue

  A. 2
  B. 3
  C. 4
  D. 5
  E. 6

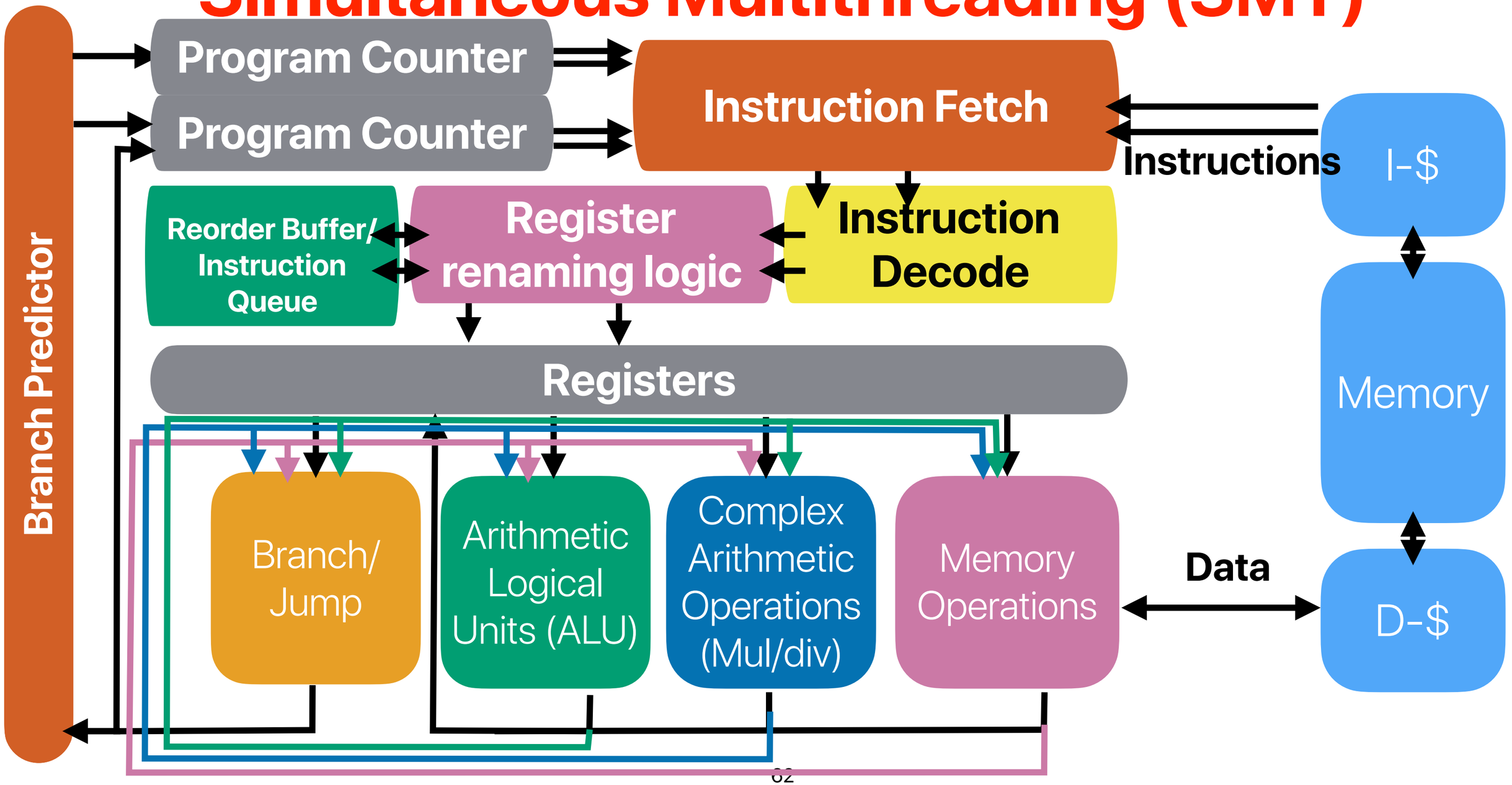
Architectural support for simultaneous multithreading

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter — **you need to have one for each context**
  - ② Register mapping tables — **you need to have one for each context**
  - ③ Physical registers — **you can share**
  - ④ ALUs — **you can share**
  - ⑤ Data cache — **you can share**
  - ⑥ Reorder buffer/Instruction Queue — **you need to indicate which context the instruction is from**
  - A. 2
  - B. 3
  - C. 4
  - D. 5
  - E. 6

How do we support two running programs in one pipeline?

- We need two program counters
- We need two sets of architectural to physical register mappings
- We do not need
  - Duplicated cache — virtually indexed, physically tagged cache already addressed that
  - Duplicated pipeline functional units — isn’t sharing the whole purpose?
  - Duplicated reorder buffer — you simply need to tag which process the instruction belongs to

Simultaneous Multithreading (SMT)



https://www.pollev.com/hungweitseng close in 1:30

Pros/Cons of SMT

- How many of the following statements about SMT is/are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can improve the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Pros/Cons of SMT

- How many of the following statements about SMT is/are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can ~~improve~~ the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

We can execute from other threads/contexts instead of the current one

hurt, b/c you are sharing resource with other threads.

We can execute from other threads/ contexts instead of the current one

b/c we're sharing the cache

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           151
Model name:                      12th Gen Intel(R) Core(TM) i3-12100F
Stepping:                        5
CPU MHz:                         3300.000
CPU max MHz:                     5500.0000
CPU min MHz:                     800.0000
BogoMIPS:                        6604.80
Virtualization:                  VT-x
L1d cache:                       192 KiB
L1i cache:                       128 KiB
```

Announcements

- **Reading Quiz 8 (last reading quiz)** due **next Tuesday** before the lecture
- **iEVAL** started and ends on 6/06/2024
  - Submit the prove of your participation in iEVAL through Gradescope
  - It can become a full credit reading quiz (it helps to amortize the penalty of another least performing one)
- **Assignment 5** is **released** due 6/09/2024
  - The same programming assignment as Assignment 3 but you need to speedup by 4x on Gradescope this time
- **Final exam**
  - 6/14 8a-11a @ **BOYHL 1471**
  - Closed book, no cheatsheet — the same rules as the midterm
  - Two questions can be used as CSMS comprehensive examine questions — one is memory-hierarchy related, and the other is OoO scheduling and code optimization

Computer Science & Engineering

203

つづく