АКОС 4

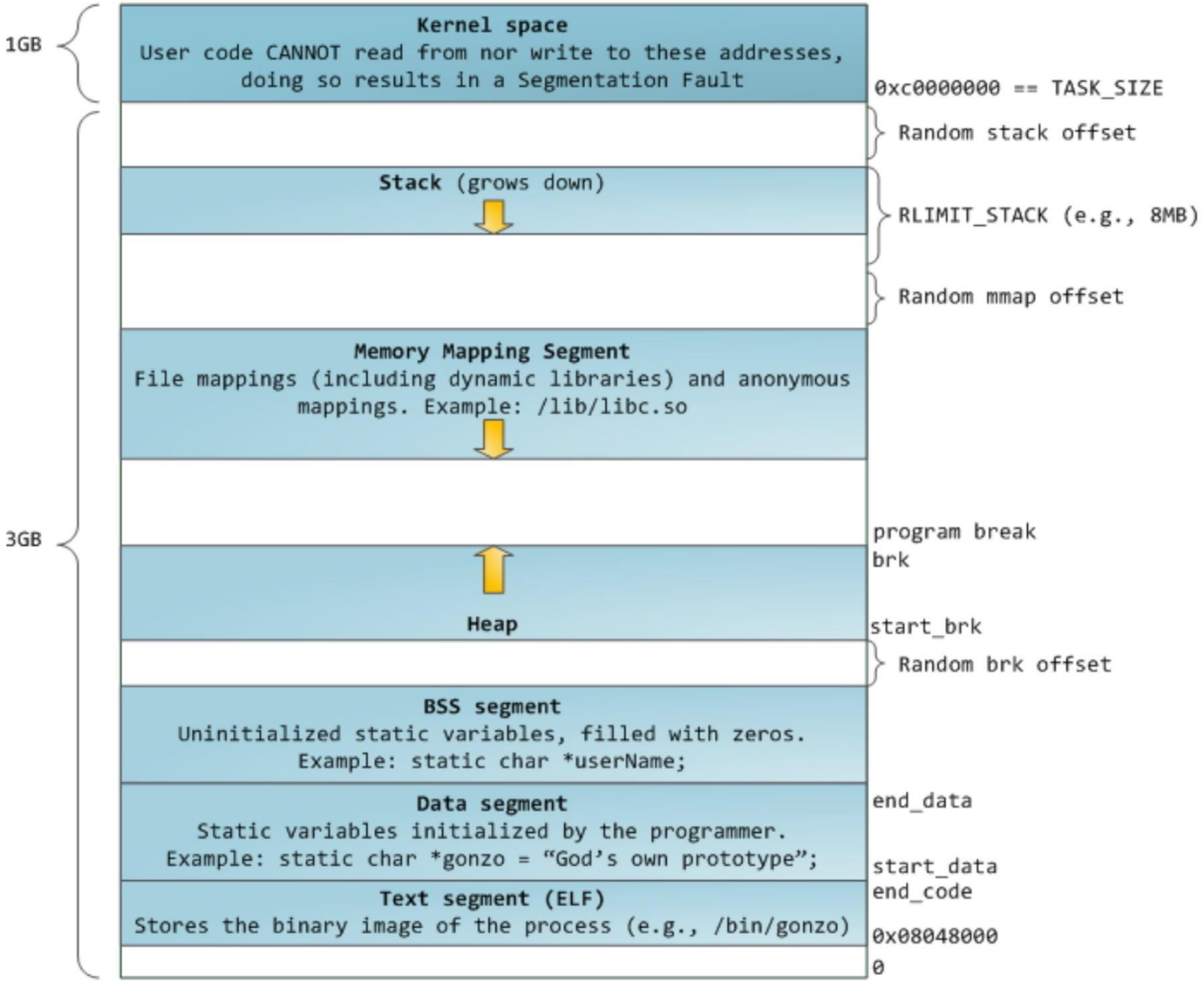
Ассемблер и память. Начало

Стадии изучения ассемблера

1. Понимание
2. Read-only
3. Read-write

1. Отрицание
2. Гнев
3. Торг
4. Депрессия
5. Принятие

Организация памяти процесса



Как посмотреть код

- objdump -M intel intel-mnemonic -d a.out

```
0000000000001216 <main>:
    1216:       f3 0f 1e fa             endbr64
    121a:       55                      push   rbp
    121b:       48 89 e5                mov    rbp,rsp
    121e:       48 83 ec 20             sub    rsp,0x20
    1222:       64 48 8b 04 25 28 00    mov    rax,QWORD PTR fs:0x28
    1229:       00 00
    122b:       48 89 45 f8             mov    QWORD PTR [rbp-0x8],rax
    122f:       31 c0                   xor    eax,eax
    1231:       c7 45 e8 00 00 00 00    mov    DWORD PTR [rbp-0x18],0x0
    1238:       c7 45 ec 01 00 00 00    mov    DWORD PTR [rbp-0x14],0x1
    123f:       c7 45 f0 02 00 00 00    mov    DWORD PTR [rbp-0x10],0x2
    1246:       48 8d 05 c3 2d 00 00    lea    rax,[rip+0x2dc3]        # 4010 <global_var>
```

Основные инструкции

Арифметические

add, sub, mul, div,

Логические

or, and, xor, inv

Регистры

mov

Прыжки

call, ret, jmp, je, ja, jle, …

Стек

push, pop

AT&T vs Intel

- objdump -M intel …
- gcc -masm=intel …
- gdb:
  - set disassembly-flavor intel
- Сравнение синтаксиса

Intex Syntax		AT&T Syntax	
mov	eax,1	movl	$1,%eax
mov	ebx,0ffh	movl	$0xff,%ebx
int	80h	int	$0x80

Intex Syntax		AT&T Syntax	
instr	dest,source	instr	source,dest
mov	eax,[ecx]	movl	(%ecx),%eax

Intex Syntax		AT&T Syntax	
mov	eax,[ebx]	movl	(%ebx),%eax
mov	eax,[ebx+3]	movl	3(%ebx),%eax

Где почитать и попробовать

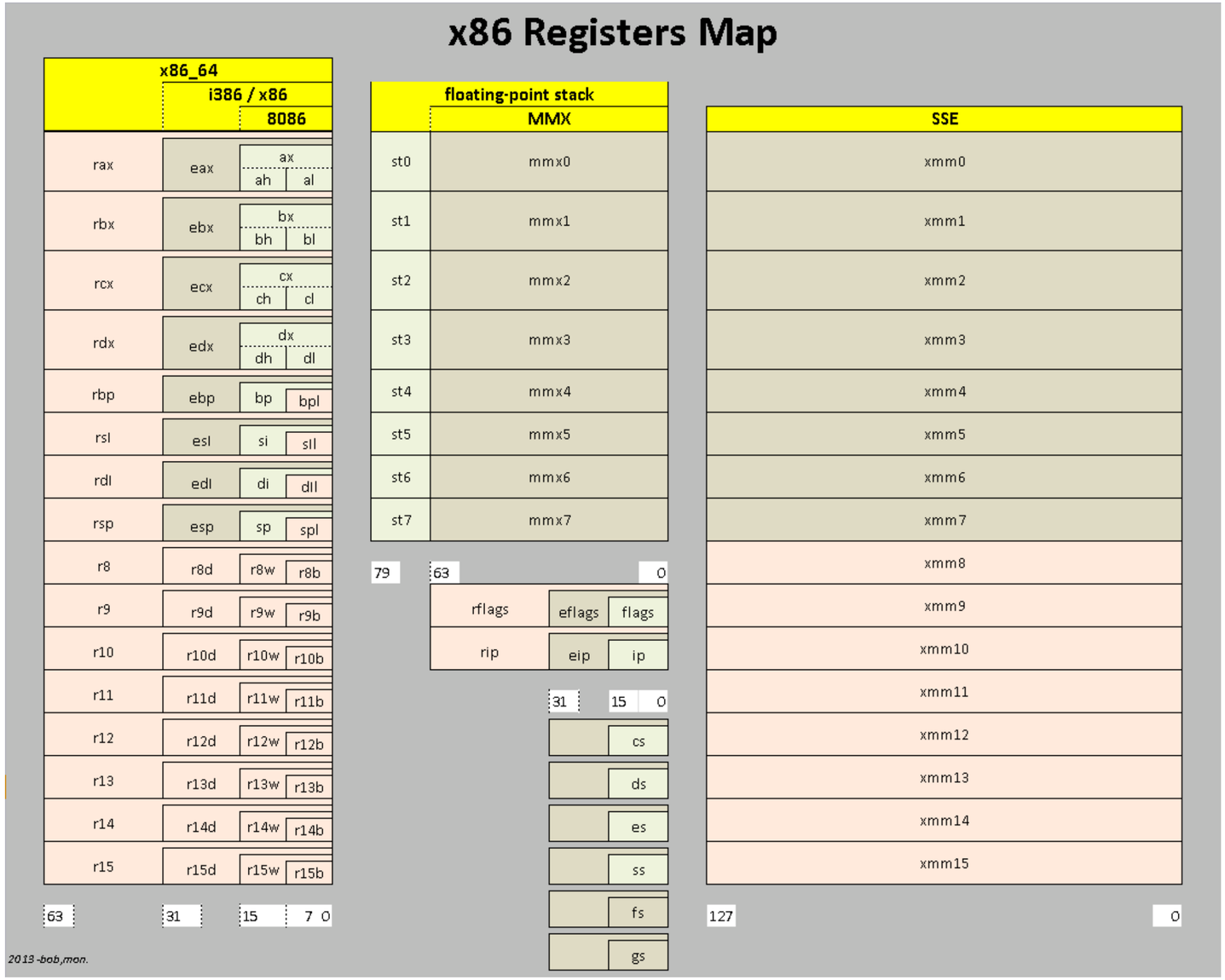
- godbolt.org
- objdump

Как писать на асме?

- Программа на асме стартер пак:
  - Объявление глобальной переменной **_start**
  - Определение метки **_start** (с неё начинается исполнение программы)
  - Завершение программы с помощью сискола exit.

Регистры

add *reg*, *reg*
add *reg*, *mem*
add *reg*, *imm*
add *mem*, *reg*
add *mem*, *imm*



Секции

- **.text** - код
- **.bss** - глобальные и статические переменные, которые проинициализированы нулём или никак
- **.data** - всё остальное
- **.rodata** - то же самое, но неизменяемое

Сравнения

```
        cmp al, n2  jne         ; сраниваем значения регистра AL и переменной n2
        not_equal mov           ; если значения не равны, переходим к метке not_equal
        eax, 0  jmp equal
not_equal:
        mov eax, 1
equal:
```

- cmp

Регистр флагов EFLAGS

Carry flag — Set if an arithmetic operation generates a carry or a borrow out of the most-significant bit of the result; cleared otherwise. This flag indicates an overflow condition for unsigned-integer arithmetic. It is also used in multiple-precision arithmetic.

Parity flag — Set if the least-significant byte of the result contains an even number of 1 bits; cleared otherwise.

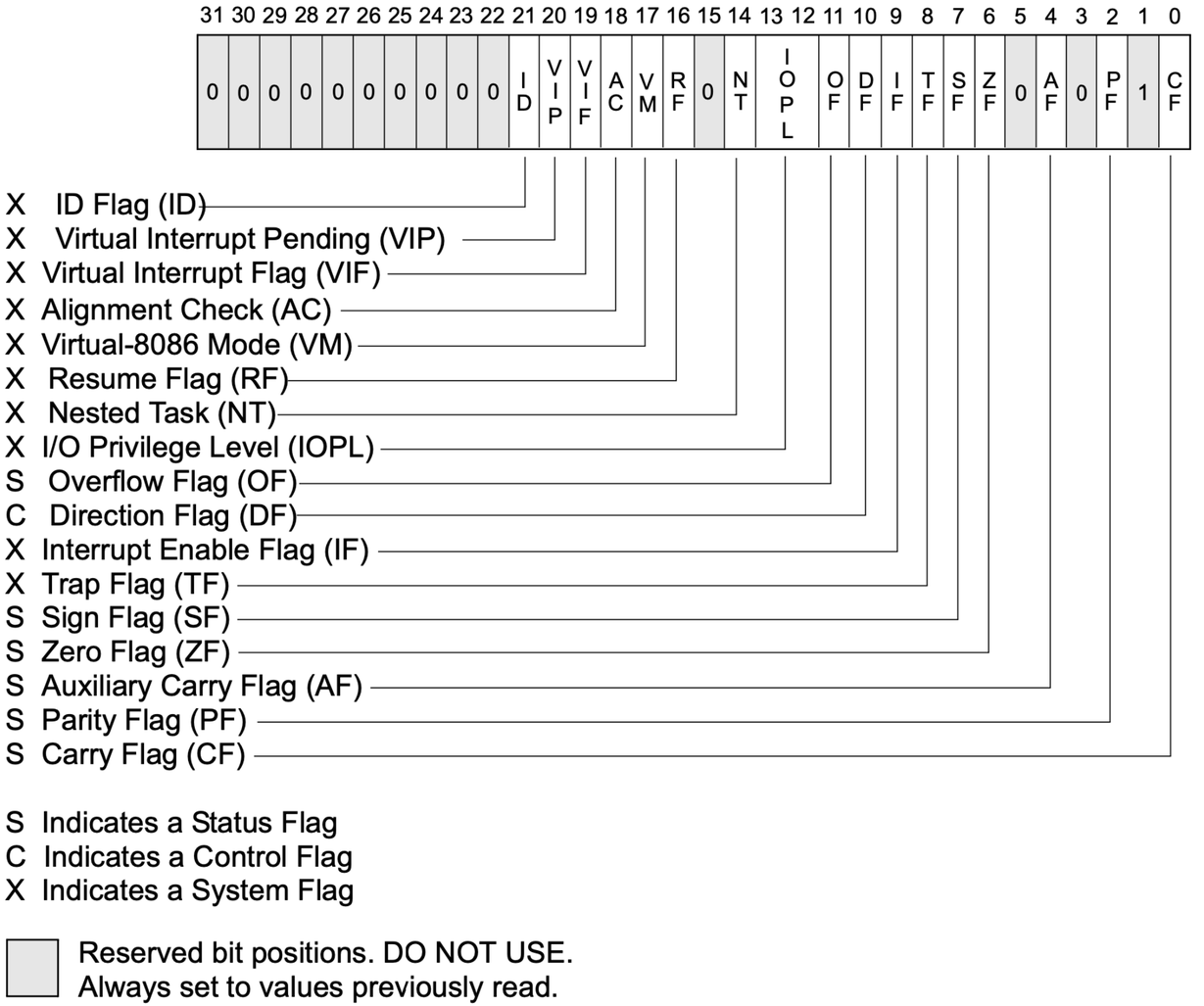
Adjust flag — Set if an arithmetic operation generates a carry or a borrow out of bit 3 of the result; cleared otherwise. This flag is used in binary-coded decimal (BCD) arithmetic.

Zero flag — Set if the result is zero; cleared otherwise.

Sign flag — Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)

Overflow flag — Set if the integer result is too large a positive number or too small a negative number (excluding the sign-bit) to fit in the destination operand; cleared otherwise. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.

- FLAGS-register



Условные переходы

```
jz      label   /* переход, если равно (нуль), ZF == 1 */
jnz     label   /* переход, если не равно (не нуль), ZF == 0 */
jc      label   /* переход, если CF == 1 */
jnc     label   /* переход, если CF == 0 */
jo      label   /* переход, если OF == 1 */
jno     label   /* переход, если OF == 0 */
jg      label   /* переход, если больше для знаковых чисел */
jge     label   /* переход, если >= для знаковых чисел */
jl      label   /* переход, если < для знаковых чисел */
jle     label   /* переход, если <= для знаковых чисел */
ja      label   /* переход, если > для беззнаковых чисел */
jae     label   /* переход, если >= (беззнаковый) */
jb      label   /* переход, если < (беззнаковый) */
jbe     label   /* переход, если <= (беззнаковый) */
```

Calling conventions

x86-64	Microsoft x64 calling convention[21]	Windows (Microsoft Visual C++, GCC, Intel C++ Compiler, Delphi), UEFI	RCX/XMM0, RDX/XMM1, R8/XMM2, R9/XMM3
	vectorcall	Windows (Microsoft Visual C++, Clang, ICC)	RCX/[XY]MM0, RDX/[XY]MM1, R8/[XY]MM2, R9/[XY]MM3 + [XY]MM4–5
	System V AMD64 ABI[28]	Solaris, Linux, BSD, macOS, OpenVMS (GCC, Intel C++ Compiler, Clang, Delphi)	RDI, RSI, RDX, RCX, R8, R9, [XYZ]MM0–7

- Разные виды calling conventions
- SystemV ABI

Register	Usage	Preserved across function calls
`%rax`	temporary register; with variable arguments passes information about the number of vector registers used; 1st return register	No
`%rbx`	callee-saved register	Yes
`%rcx`	used to pass 4th integer argument to functions	No
`%rdx`	used to pass 3rd argument to functions; 2nd return register	No
`%rsp`	stack pointer	Yes
`%rbp`	callee-saved register; optionally used as frame pointer	Yes
`%rsi`	used to pass 2nd argument to functions	No
`%rdi`	used to pass 1st argument to functions	No
`%r8`	used to pass 5th argument to functions	No
`%r9`	used to pass 6th argument to functions	No
`%r10`	temporary register, used for passing a function's static chain pointer	No
`%r11`	temporary register	No
`%r12-r14`	callee-saved registers	Yes
`%r15`	callee-saved register; optionally used as GOT base pointer	Yes
`%xmm0–%xmm1`	used to pass and return floating point arguments	No
`%xmm2–%xmm7`	used to pass floating point arguments	No
`%xmm8–%xmm15`	temporary registers	No
`%mmx0–%mmx7`	temporary registers	No
`%st0,%st1`	temporary registers; used to return `long double` arguments	No
`%st2–%st7`	temporary registers	No
`%fs`	Reserved for system (as thread specific data register)	No
`mxcsr`	SSE2 control and status word	partial
`x87 SW`	x87 status word	No
`x87 CW`	x87 control word	Yes

gdb features

- i(nfo) r(egister) (rax)
- p $eflags
- x /nfu addr (z.B. x/3uh 0x54320)
- ~/.gdbinit – настройки gdb
- set disassembly-flavor intel

Форматы чтения

Intrinsics

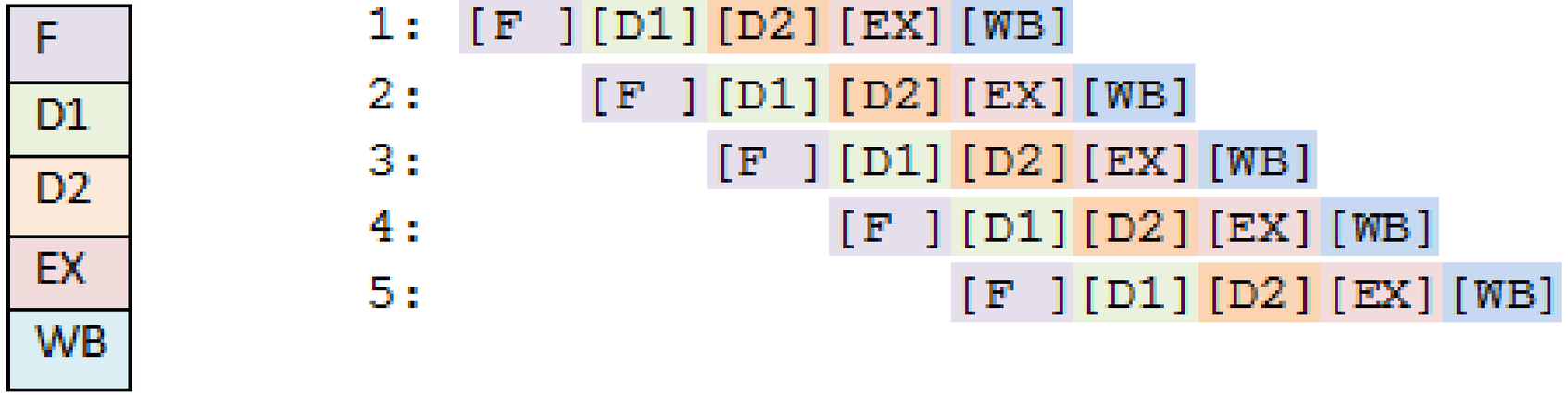
```
__m256 r1 = _mm256_load_ps(a + i);
__m256 r2 = _mm256_load_ps(b + i);
__m256 r3 = _mm256_add_ps(r1, r2);
```

Гайд по интринсикам

Что посмотреть?

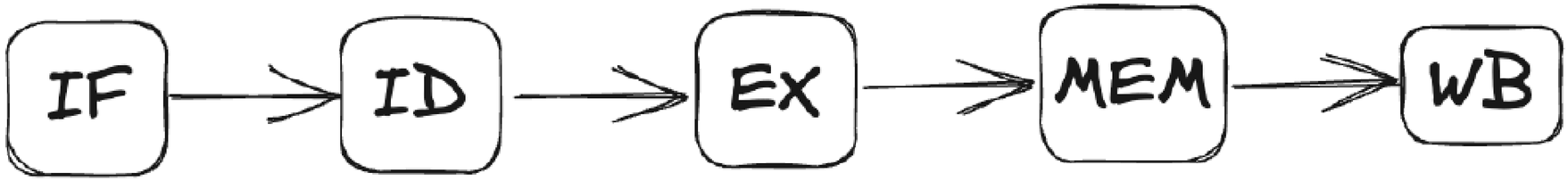
- Intel Architecture Manual vol.1
- Intel Architecture Manual vol. 2
- nasm tutorial

Pipelining



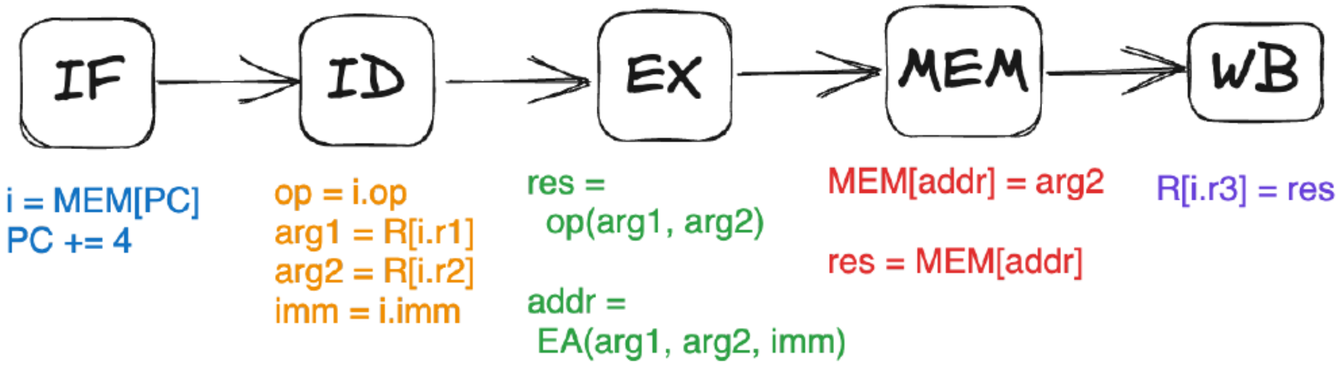
i486 pipeline

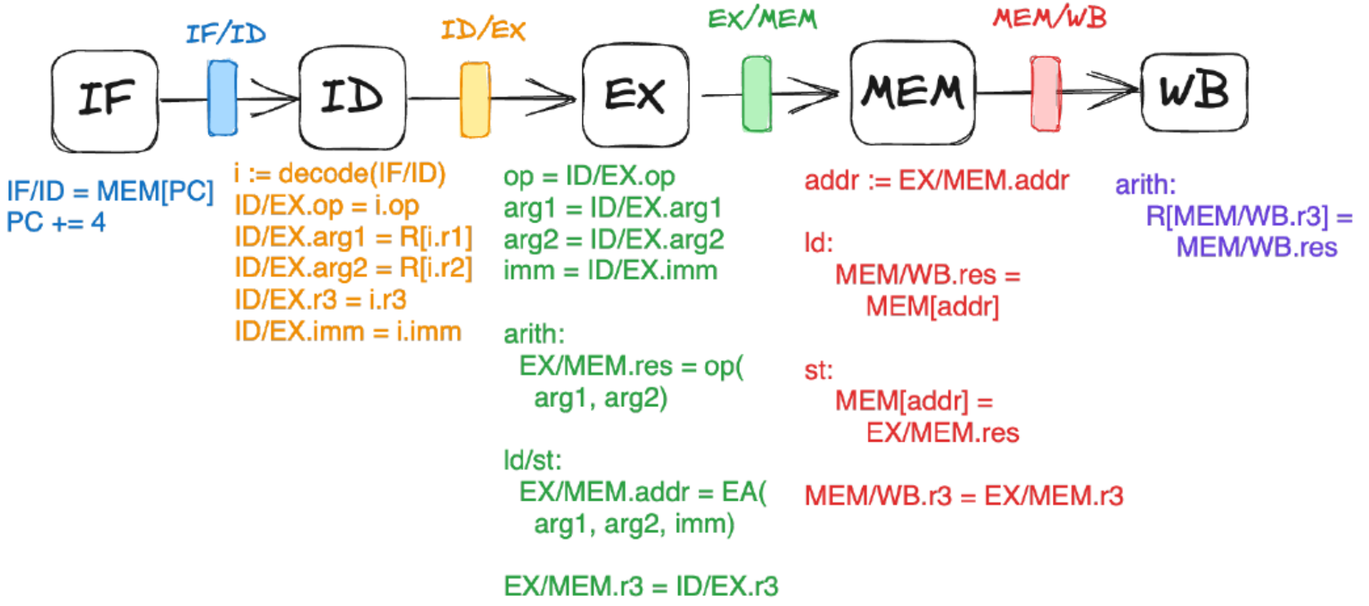
Pipelining



IF - Instruction Fetch
ID - Instruction Decode
EX - Execution
MEM - Memory access: Load/Store
WB - Write-back

```
add r1, r2, r3
ld r1, 8(r2)
st 8(r2), r3
```

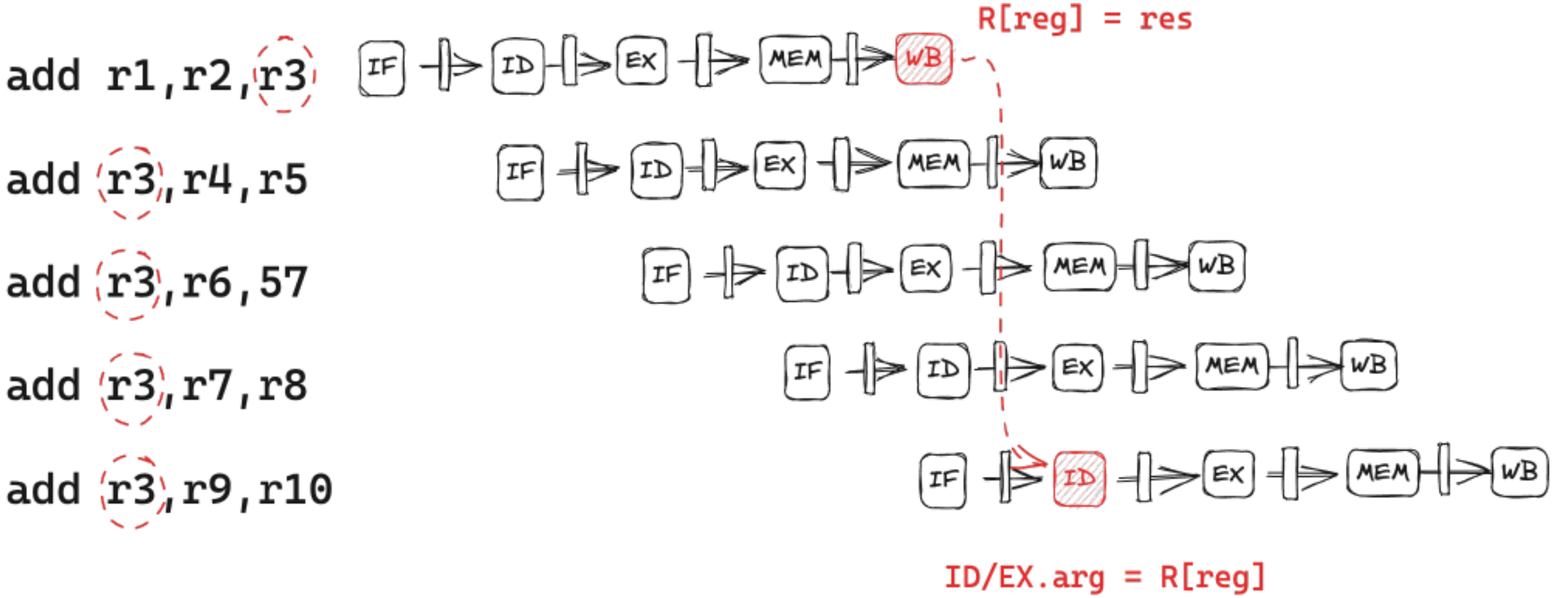




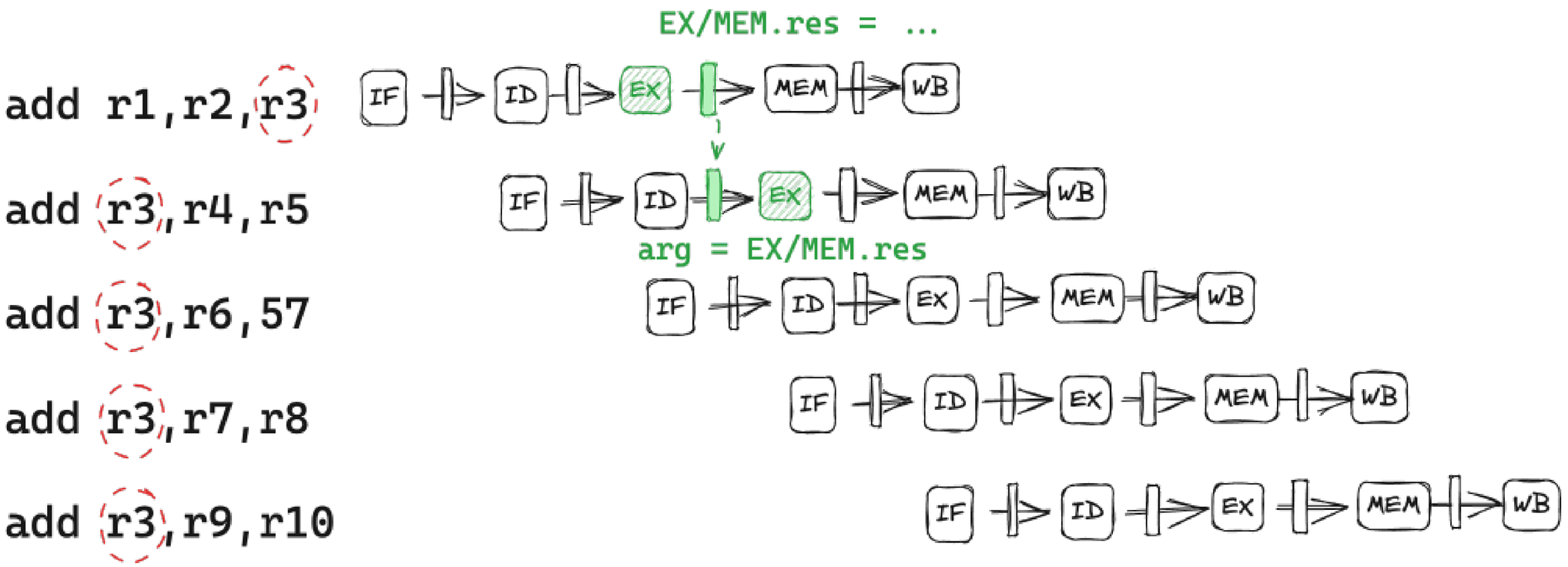
Pipeline stall/bubble (простой конвеера)

```
xor a, b
xor b, a
xor a, b
```

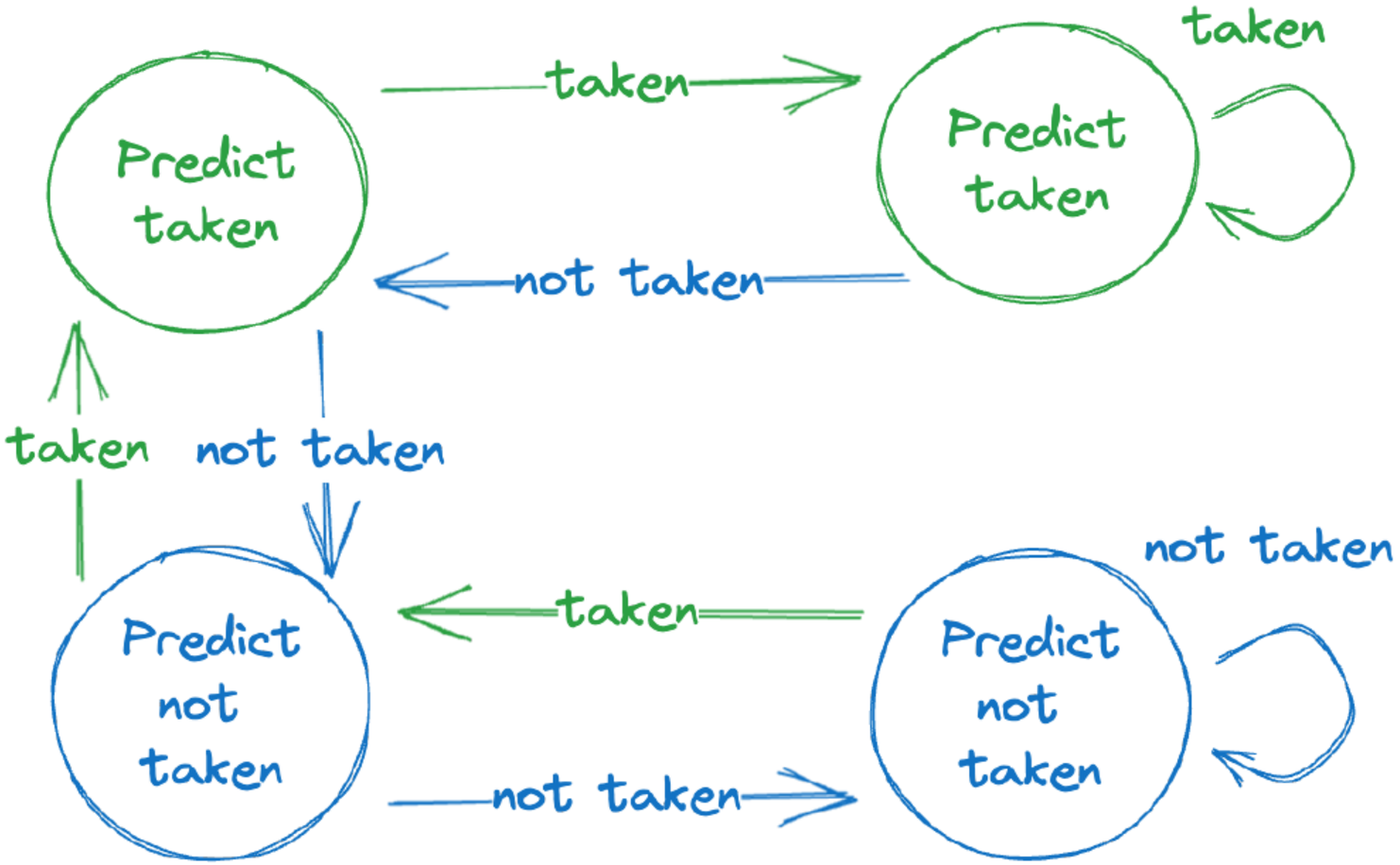
Bypass


add r1,r2,r3
add r3,r4,r5
add r3,r6,57
add r3,r7,r8
add r3,r9,r10
IF
ID
EX
MEM
WB
R[reg] = res
ID/EX.arg = R[reg]

Bypass


EX/MEM.res = ...
add r1,r2,r3
add r3,r4,r5
arg = EX/MEM.res
add r3,r6,57
add r3,r7,r8
add r3,r9,r10
IF
ID
EX
MEM
WB

Simple dynamic branch predictor



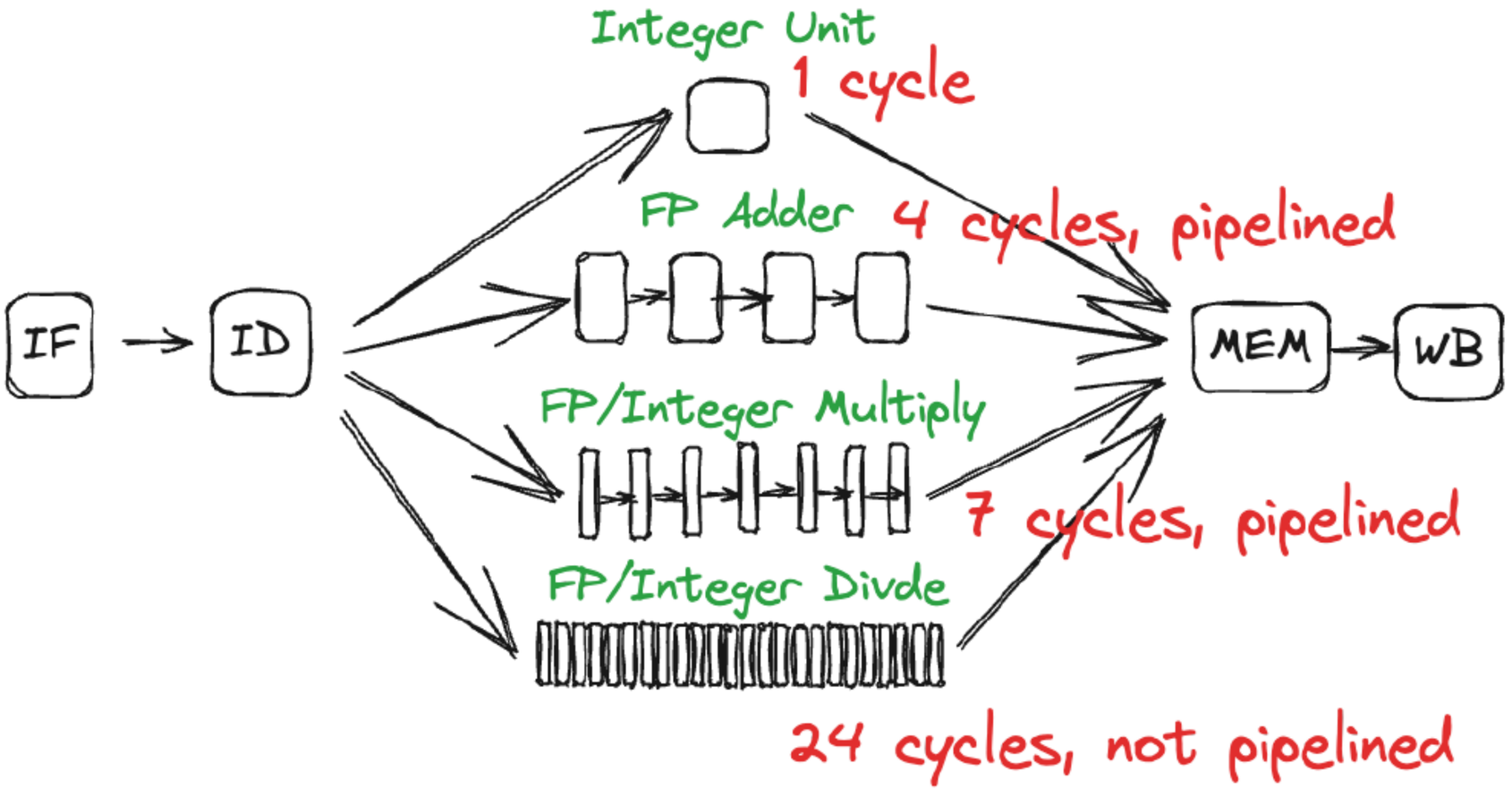
Branches

__builtin_expect(long expr, long expected) – выполнится ли условие – expected = 0 – не выполнится
[[likely]] / [[unlikely]]

```
if(__builtin_expect(b(true), 1))
{
    std::cout << " main true" << std::endl;
}
```

Необходимо включить оптимизации хотя бы –O1
Статья на хабре
Перформанс

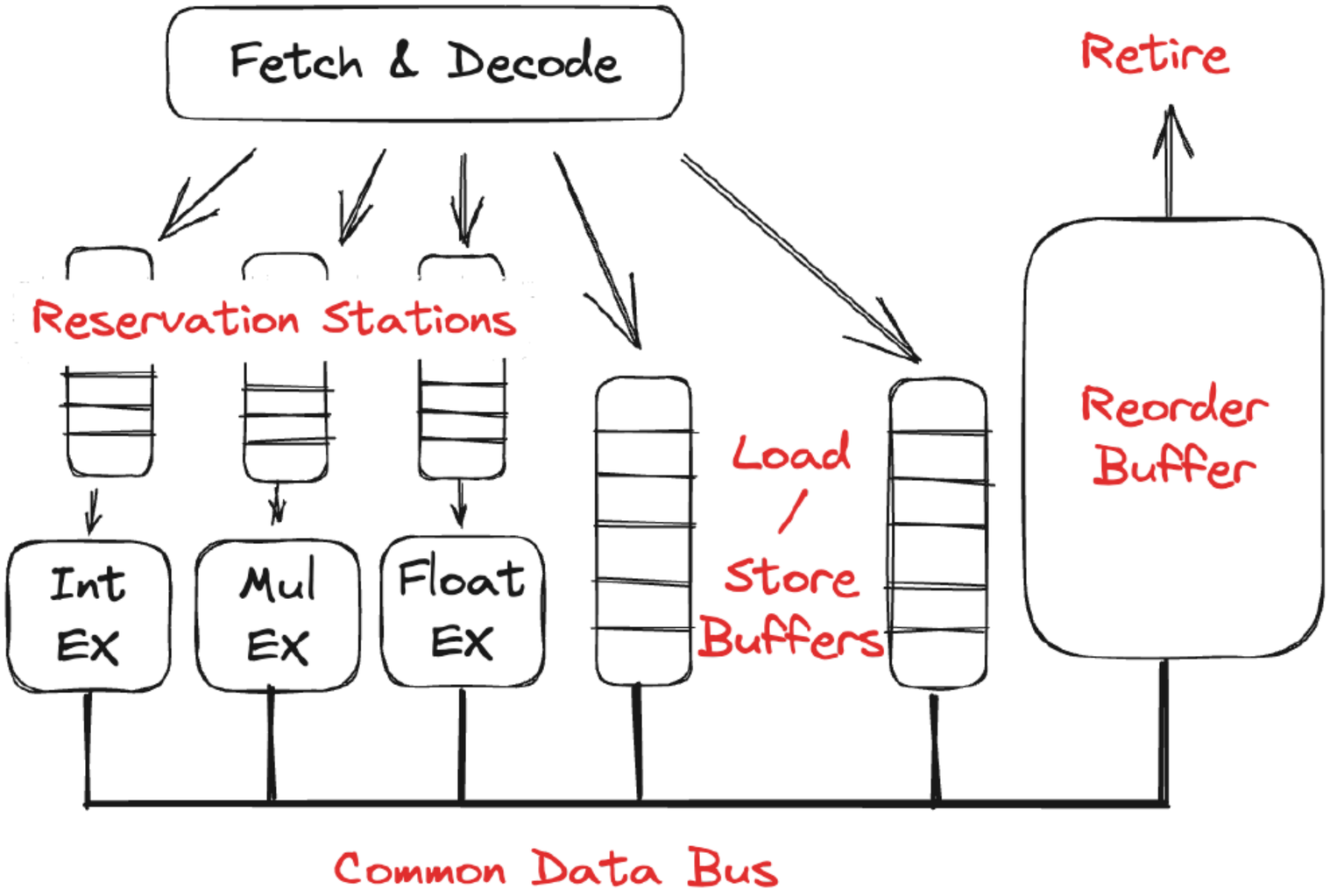
Execution units



Hazards

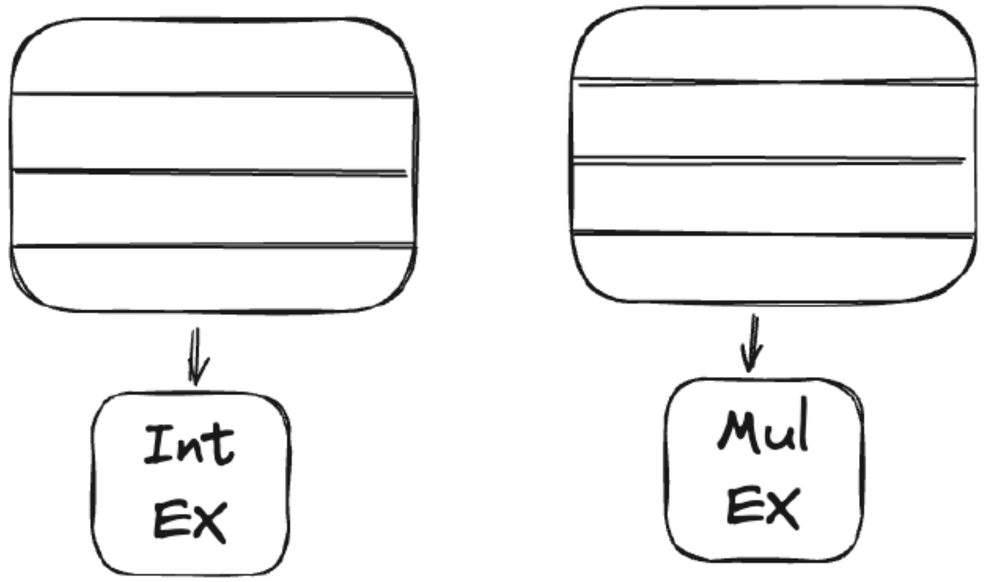
- **Data hazard**. Конфликты по данным
  - **Read after write.** Чтение после записи должно вернуть что записали.
  - **Write after read.** Запись после чтения не должна быть видна при чтении.
  - **Write after write.** Запись после записи должна перетереть старое значение.
- **Control hazard.** Соблюдение семантики ветвлений
- **Structural hazard.** Конфликты за ресурсы: execution unit занят

Tamasulo's algorithm

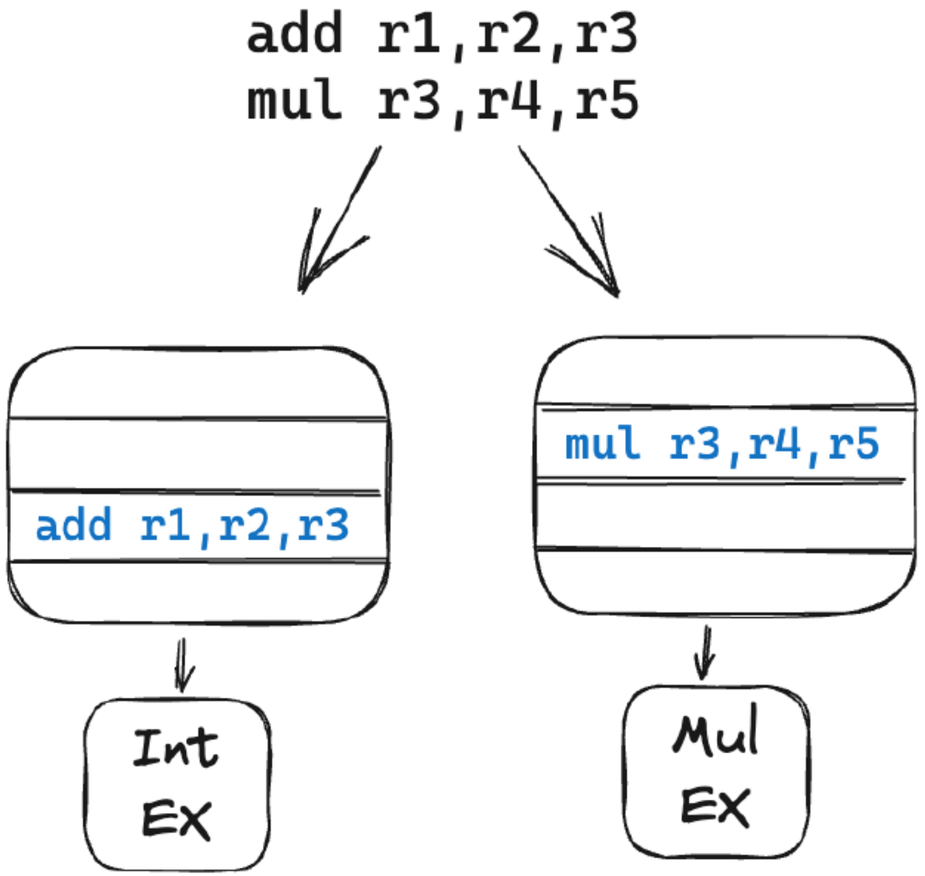


Reservation stations

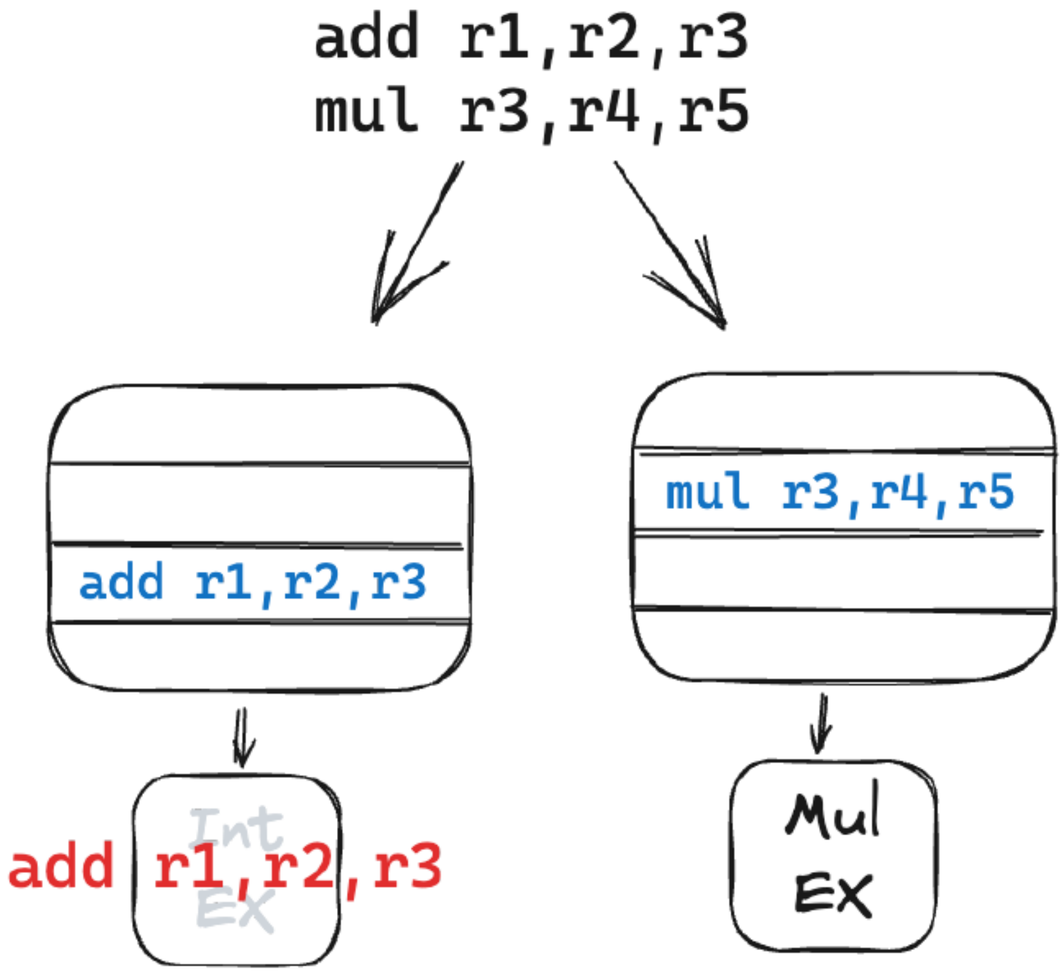
```
add r1,r2,r3
mul r3,r4,r5
```



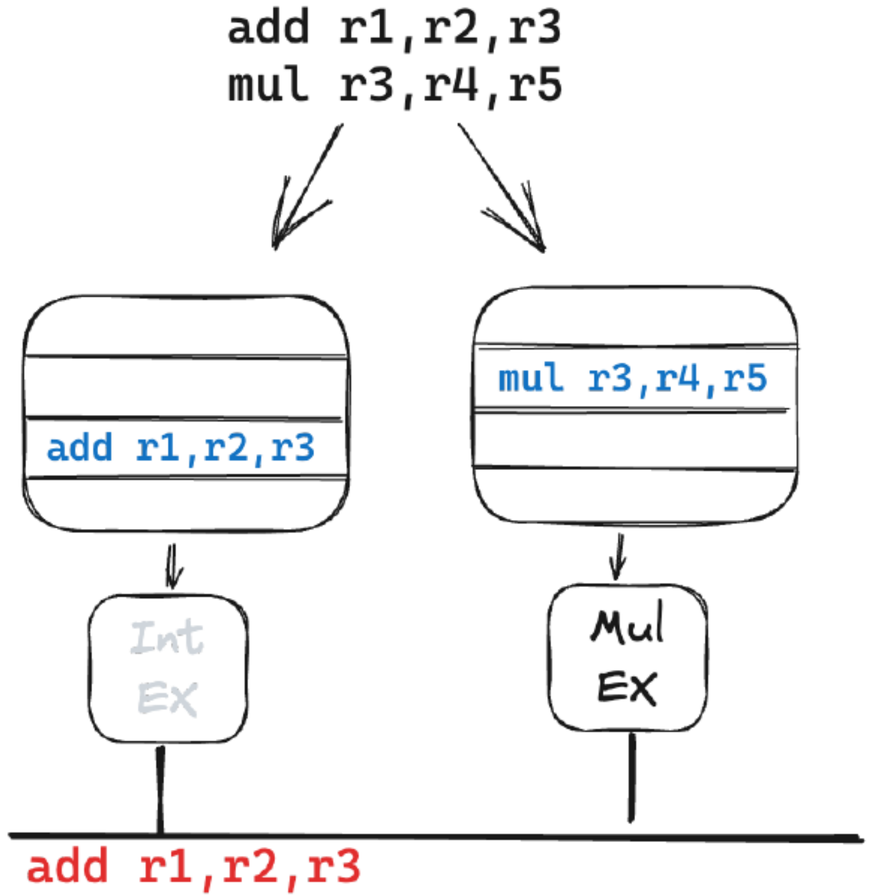
Reservation stations



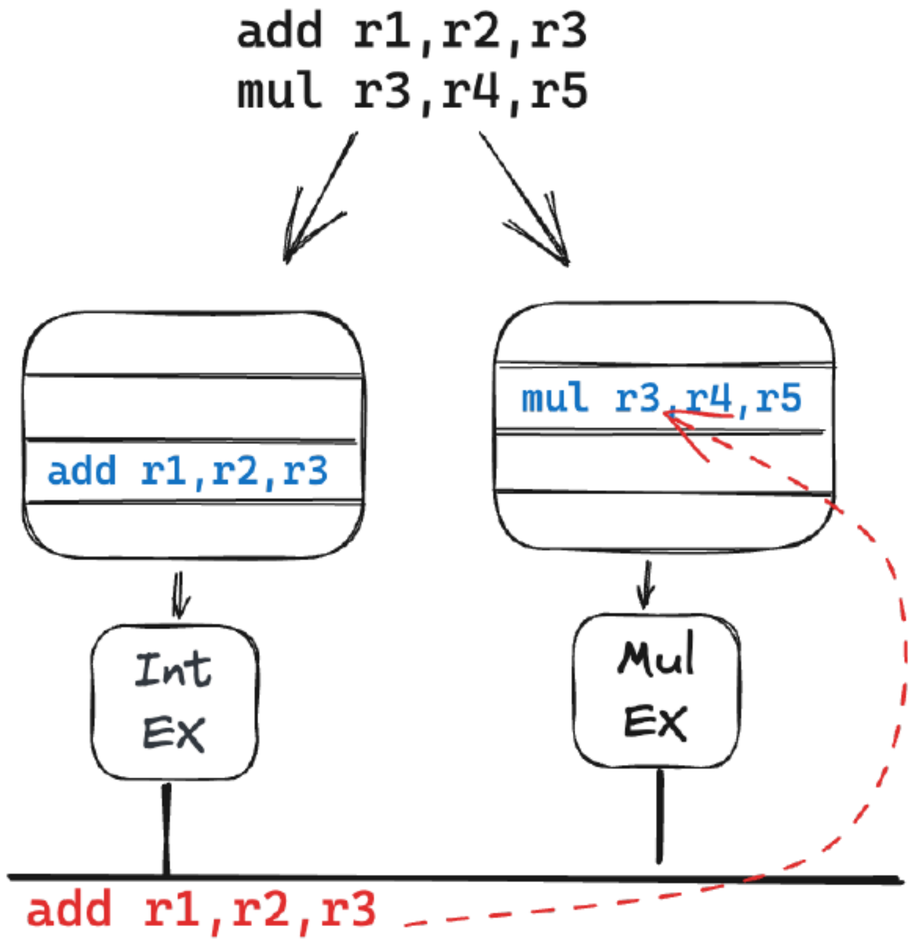
Reservation stations



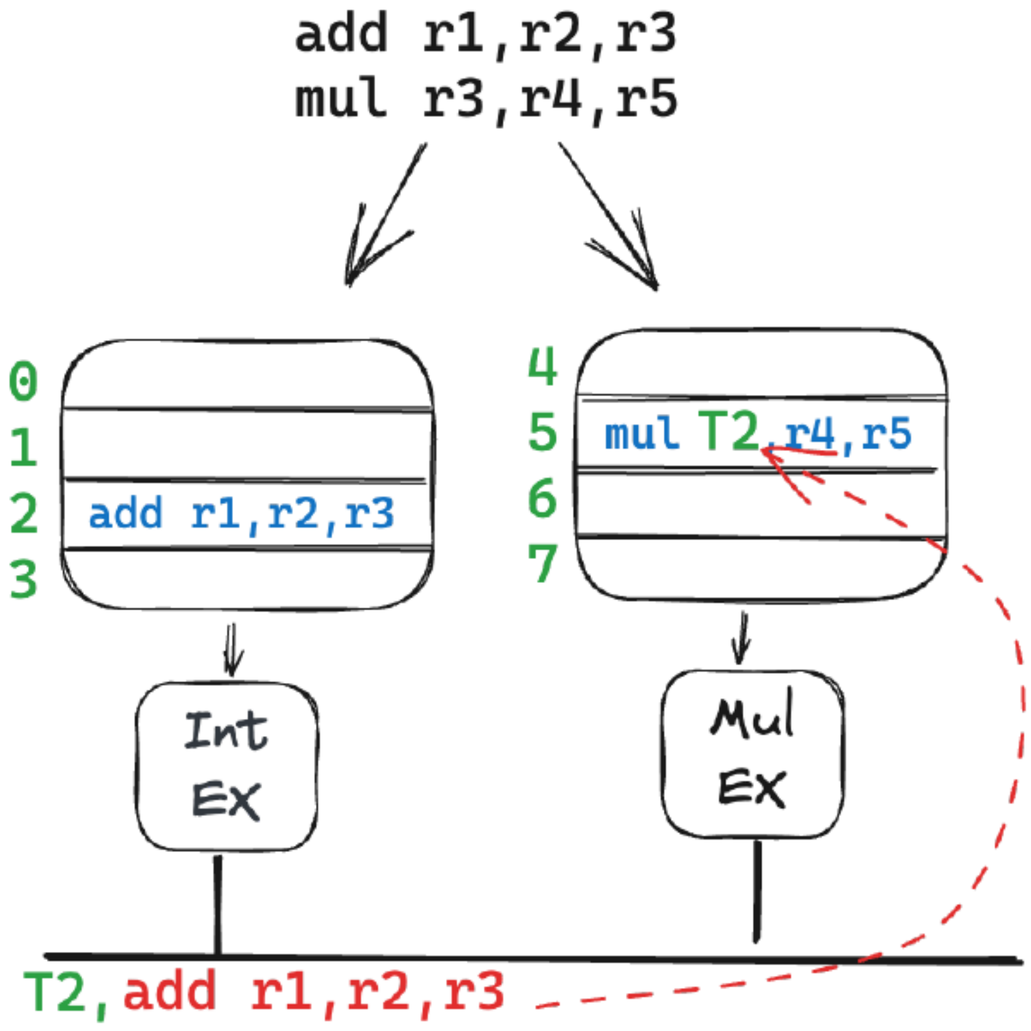
Common Data Bus



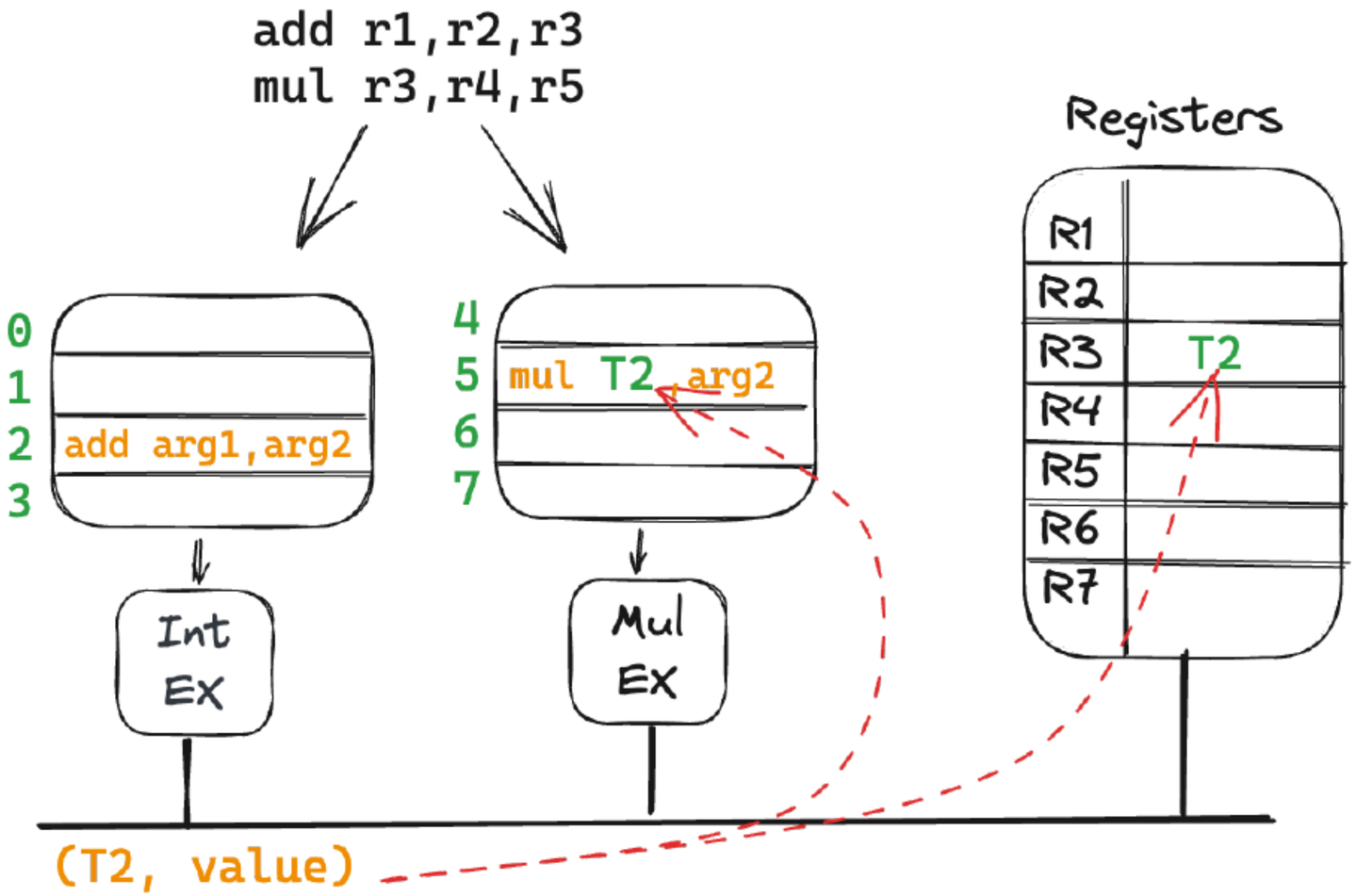
Common Data Bus



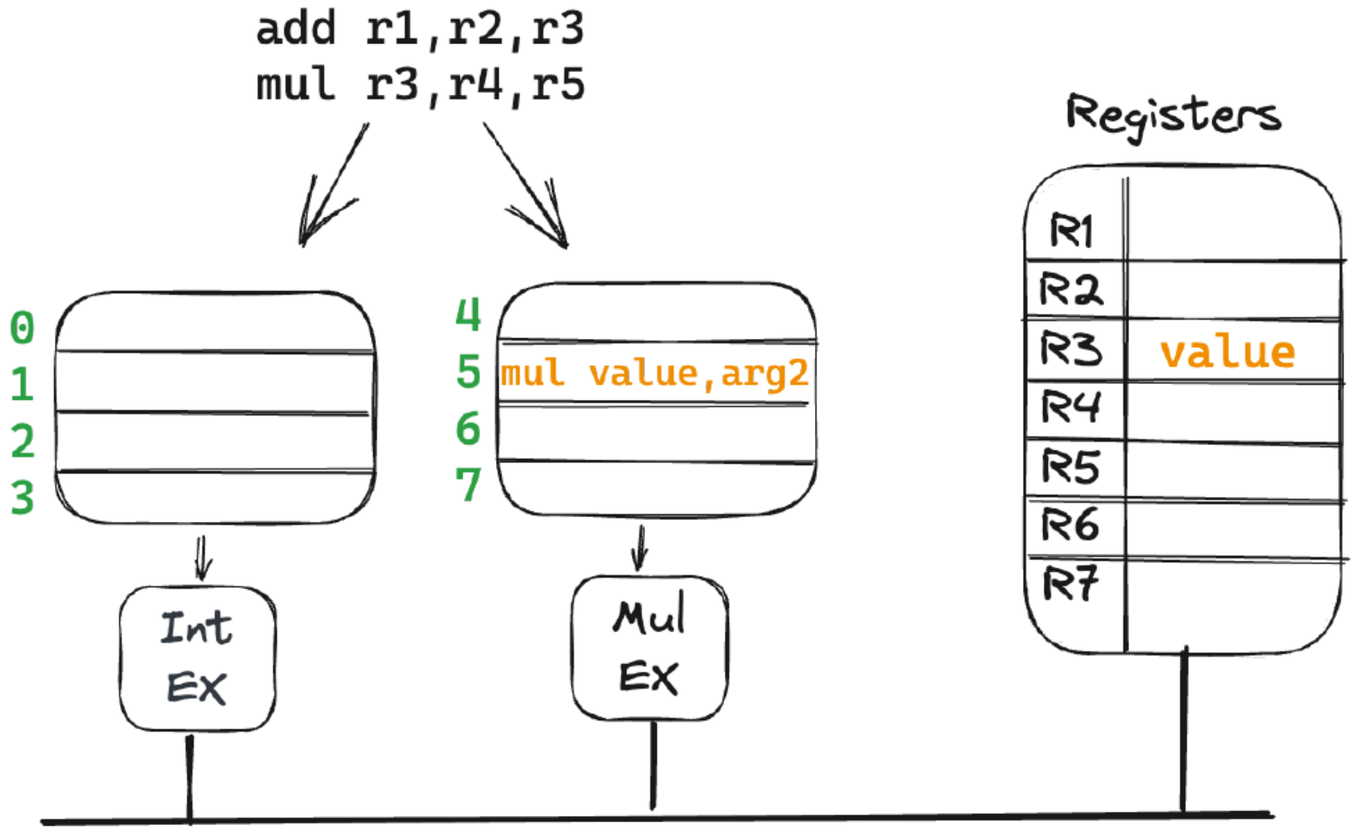
Tags


add r1,r2,r3
mul r3,r4,r5
0
1
2
3
add r1,r2,r3
4
5
6
7
mul T2,r4,r5
Int
Ex
Mul
Ex
T2,add r1,r2,r3

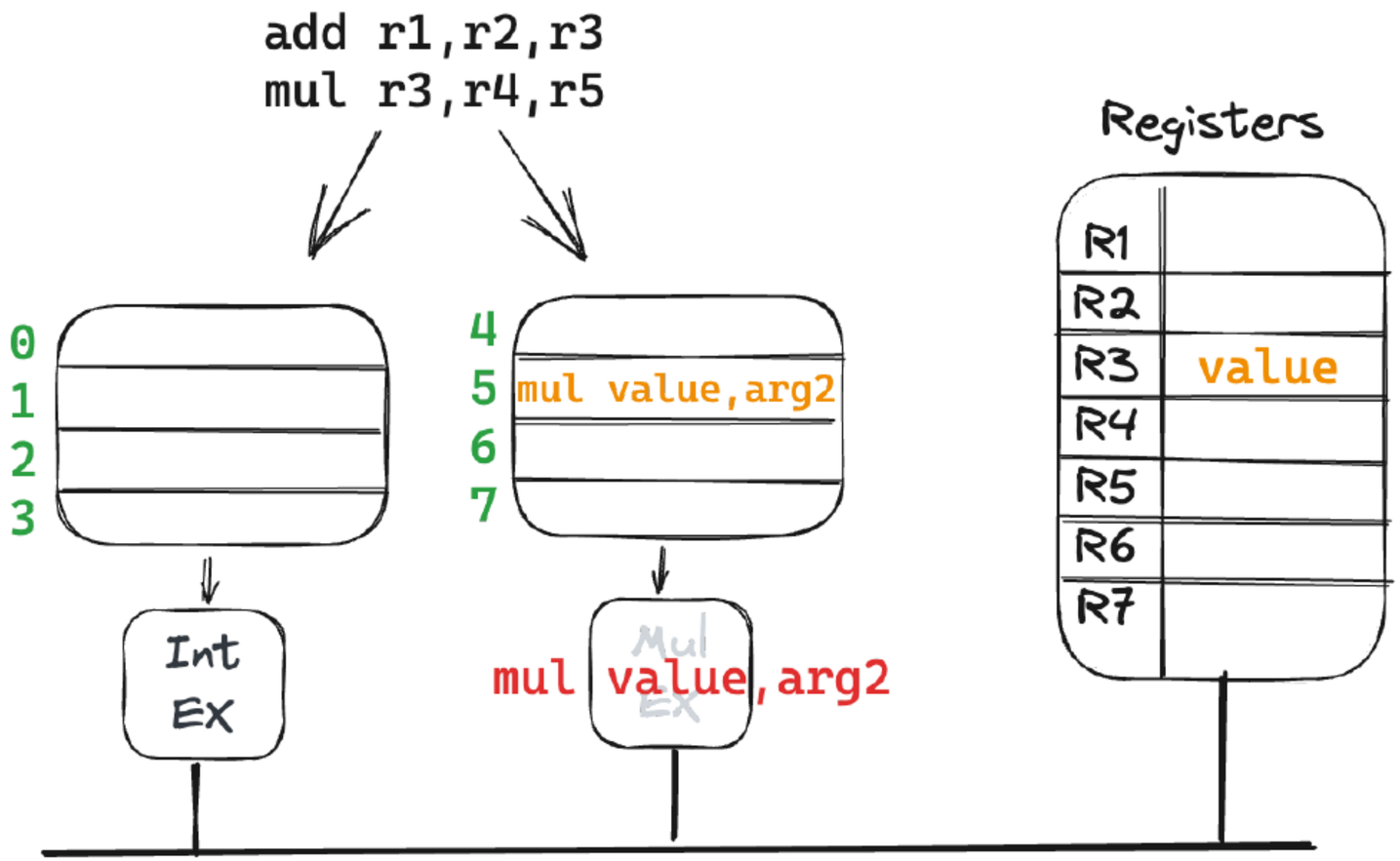
Tags


add r1,r2,r3
mul r3,r4,r5
Registers
R1
R2
R3
T2
R4
R5
R6
R7
0
1
2
3
add arg1,arg2
4
5
mul T2, arg2
6
7
Int
EX
Mul
EX
(T2, value)

Tags


add r1,r2,r3
mul r3,r4,r5
0
1
2
3
4
5
6
7
mul value,arg2
Int EX
Mul EX
Registers
R1
R2
R3
value
R4
R5
R6
R7

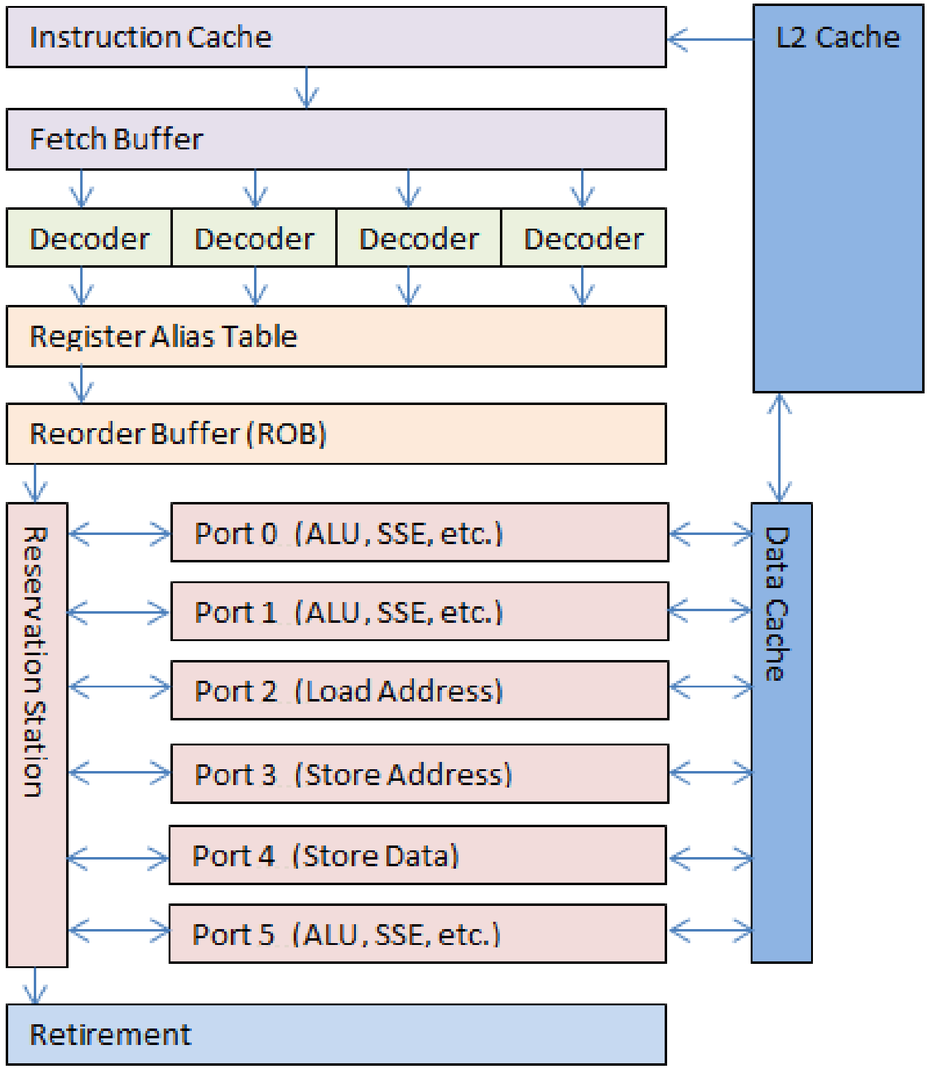
Tags


add r1,r2,r3
mul r3,r4,r5
Registers
R1
R2
R3
value
R4
R5
R6
R7
0
1
2
3
4
5
mul value,arg2
6
7
Int
EX
Mul
EX
mul value,arg2

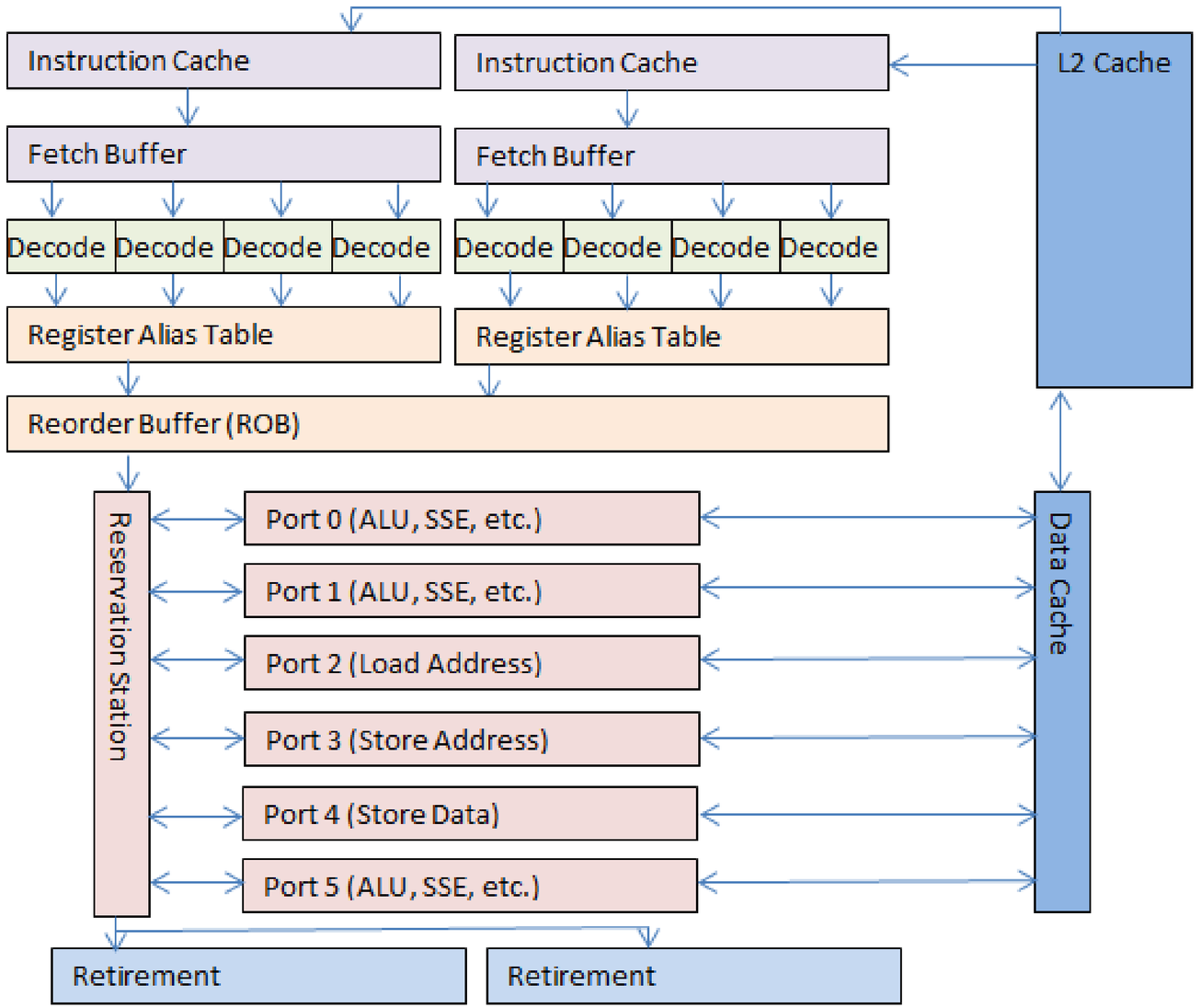
Итого

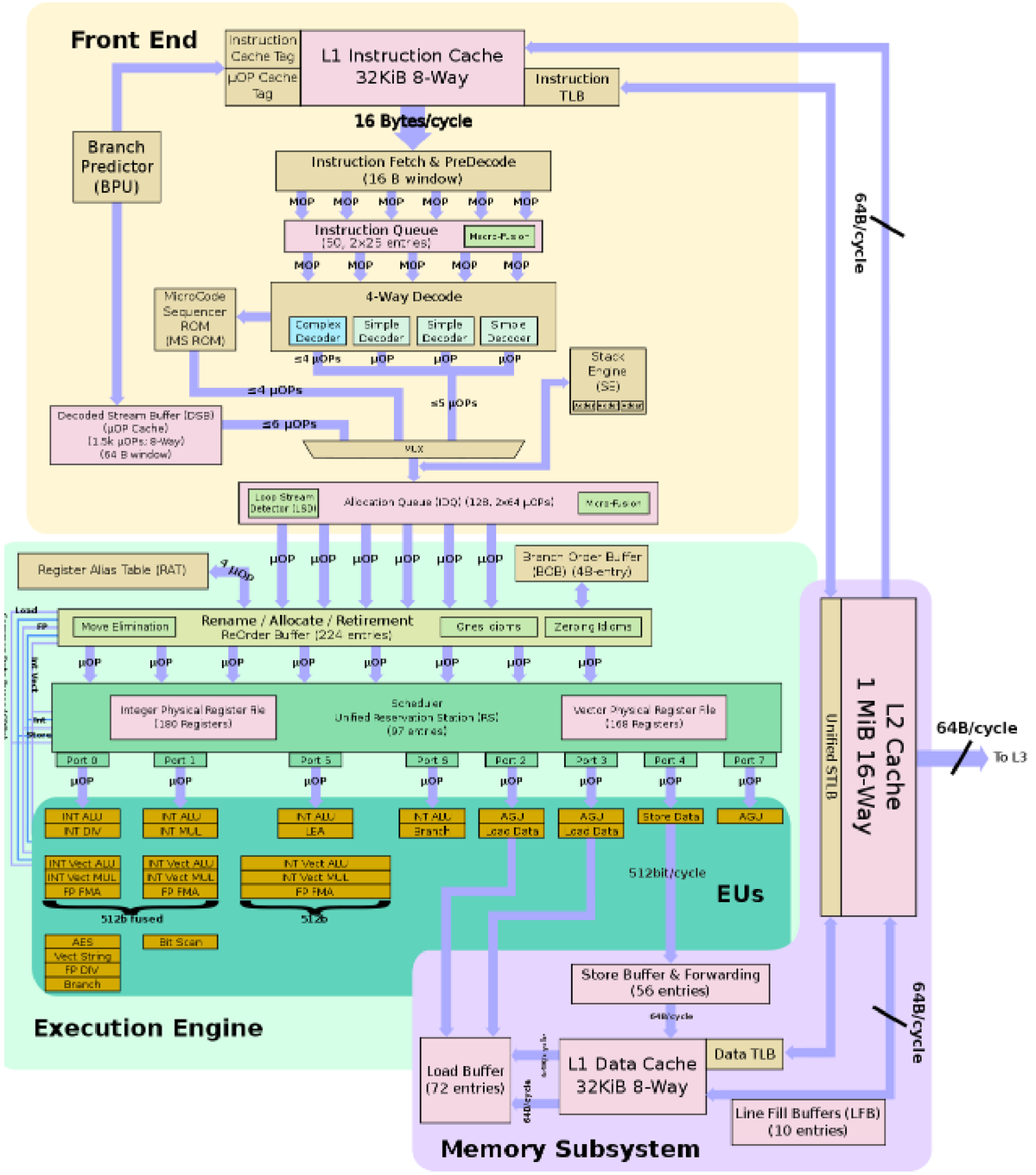
- Сохранение семантики (RAW, WAR, WAW) благодаря register renaming
- Обработка исключений - с помощью ROB
- Спекулятивное выполнение - с помощью ROB

Out-of-order


Instruction Cache
L2 Cache
Fetch Buffer
Decoder
Decoder
Decoder
Decoder
Register Alias Table
Reorder Buffer (ROB)
Reservation Station
Port 0 (ALU, SSE, etc.)
Port 1 (ALU, SSE, etc.)
Port 2 (Load Address)
Port 3 (Store Address)
Port 4 (Store Data)
Port 5 (ALU, SSE, etc.)
Data Cache
Retirement

Out-of-order + Hyper-Threading




Front End
Instruction Cache Tag
µOP Cache Tag
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP
Instruction Queue (50, 2x25 entries)
Micro-Fusion
MicroCode Sequencer ROM (MS ROM)
4-Way Decode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
≤4 µOPs
µOP
Stack Engine (SE)
sub
ecx
sub
≤4 µOPs
≤5 µOPs
Decoded Stream Buffer (DSB) (µOP Cache) (1.5k µOPs; 8-Way) (64 B window)
≤6 µOPs
VLX
Loop Stream Detector (LSD)
Allocation Queue (IDQ) (128, 2x64 µOPs)
Micro-Fusion
64B/cycle
Register Alias Table (RAT)
7 µOPs
µOP
Branch Order Buffer (BOB) (48-entry)
Load
FP
Move Elimination
Rename / Allocate / Retirement
ReOrder Buffer (224 entries)
Gen. Idioms
Zeroing Idioms
Integer Physical Register File (180 Registers)
Scheduler
Unified Reservation Station (RS)
(97 entries)
Vector Physical Register File (168 Registers)
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
INT ALU
INT DIV
INT ALU
INT MUL
INT ALU
LEA
INT ALU
Branch
AGU
Load Data
AGU
Load Data
Store Data
AGU
INT Vect ALU
INT Vect MUL
FP FMA
INT Vect ALU
INT Vect MUL
FP FMA
INT Vect ALU
INT Vect MUL
FP FMA
512b fused
512b
512bit/cycle
EUs
AES
Vect String
FP DIV
Branch
Bit Scan
Execution Engine
L2 Cache
1 MiB 16-Way
Unified STLB
64B/cycle
To L3
Store Buffer & Forwarding (56 entries)
64B/cycle
Load Buffer (72 entries)
L1 Data Cache 32KiB 8-Way
Data TLB
64B/cycle
64B/cycle
Line Fill Buffers (LFB) (10 entries)
Memory Subsystem

Прерывания

- **Асинхронные** или **внешние** (аппаратные) — события, которые исходят от внешних источников и могут произойти в любой произвольный момент. Факт возникновения в системе такого прерывания трактуется как запрос на прерывание (англ. Interrupt request, IRQ)
  - I/O Device service-request, таймер, hardware/power failure
- **Синхронные** или **внутренние (traps и exceptions)** — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода
  - Undefined opcode, arithmetic overflow, FPU exception, misaligned memory
  - Virtual memory exceptions: page faults, TLB misses, protection violations
  - Traps (программные) — инициируются исполнением специальной инструкции в коде программы (system calls)

Примеры

Table 45: List of MCAUSE Exception Codes

INT	EC	Description
0	0	Custom breakpoint with entering Debug Mode
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	Reserved
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	Custom TLB miss
0	15	Store/AMO page fault
0	63..16	Reserved

Floating Point Exceptions

Table 7: Structure of FCSR register

Bit number	Access	Description
0	RW	FFLAGS[0] - Inexact result flag, NX
1	RW	FFLAGS[1] - Underflow flag, UF
2	RW	FFLAGS[2] - Overflow flag, OF
3	RW	FFLAGS[3] - Divide by zero flag, DZ
4	RW	FFLAGS[4] - Invalid operation flag, NV
7..5	RW	FRM[2..0] - Rounding mode
63..8	RZ	Reserved

Типы прерываний

- **Маскируемые** — прерывания, которые можно запрещать установкой соответствующих битов в регистре маскирования прерываний (в x86-процессорах — сбросом флага IF в регистре флагов)

- **Немаскируемые** (англ. Non maskable interrupt, NMI) — обрабатываются всегда, независимо от запретов на другие прерывания

Обработка прерываний

- Текущая программа остановлена на команде I_i, все предыдущие команды завершены до I_{i-1} включительно (*точное прерывание)*

- Значение PC команды I_i сохранено в специальный регистр (EPC)

- Новые прерывания замаскированы, управление передается обработчику соответствующего прерывания (Interrupt Handler), исполняющемуся в kernel mode

Обработчик прерываний

- Сохранение EPC до включения прерываний (отложенные прерывания) $\Rightarrow$
  - команда записи EPC в GPRs
  - до завершения записи EPC новые прерывания должны быть замаскированы
- Чтение регистра статуса (Cause register) для определения причины прерывания
- Специальная команда перехода SRET (*return-from-supervisor*)
  - включение прерываний
  - возвращение в пользовательский режим
  - восстановление архитектурного состояния

Синхронное прерывание

- Возникает во время исполнения *определенной команды*
- В общем случае команда не может быть завершена и должна быть повторно исполнена после обработки прерывания
- В случае системных вызовов команда (system call) считается исполненной
  - специальная команда перехода в privileged kernel mode

Что почитать

Intel Skylake