Embedded SoC

Term Project

Designing General Purpose Parallel Compute Unit

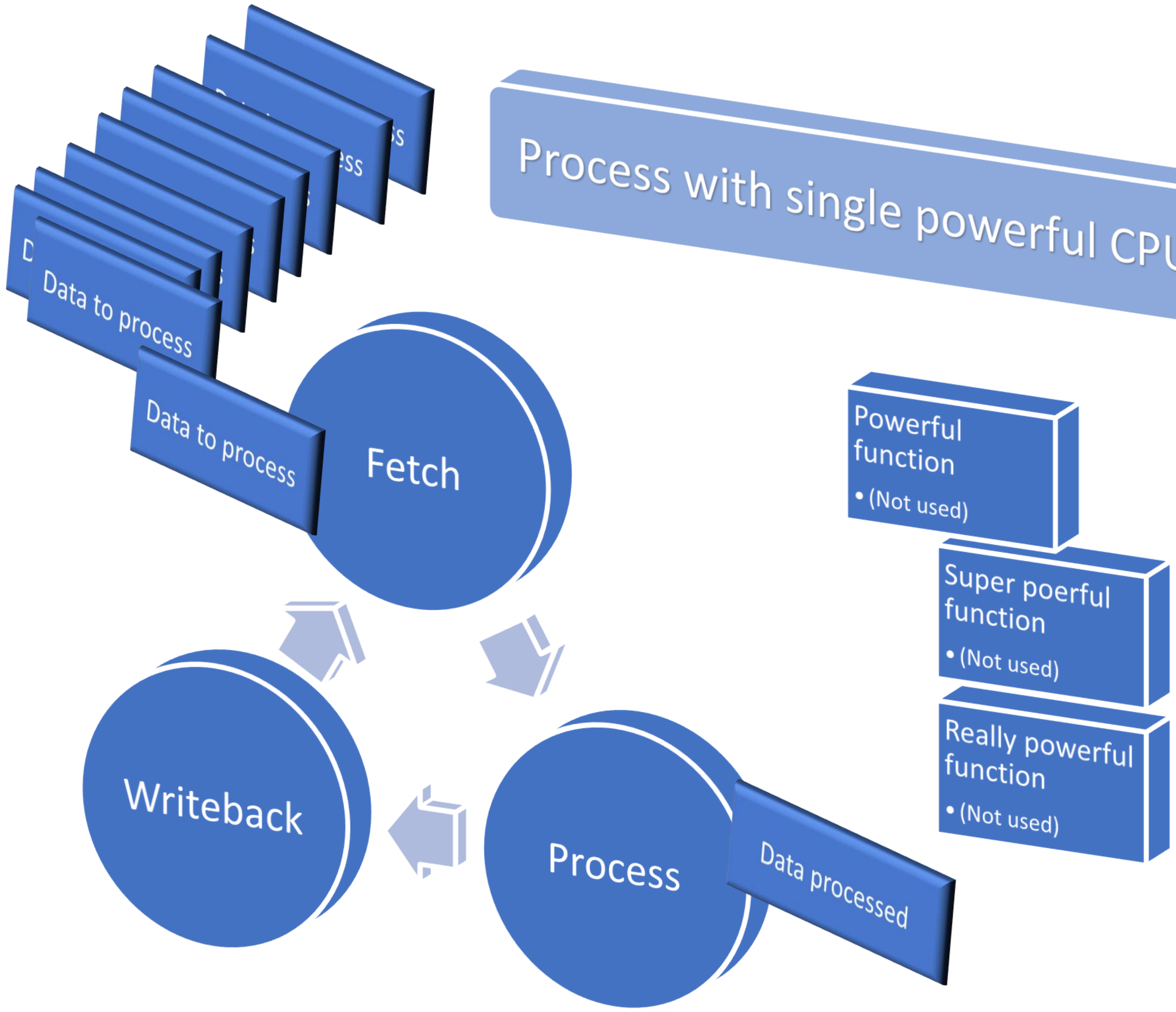
범용 병렬 계산기 설계

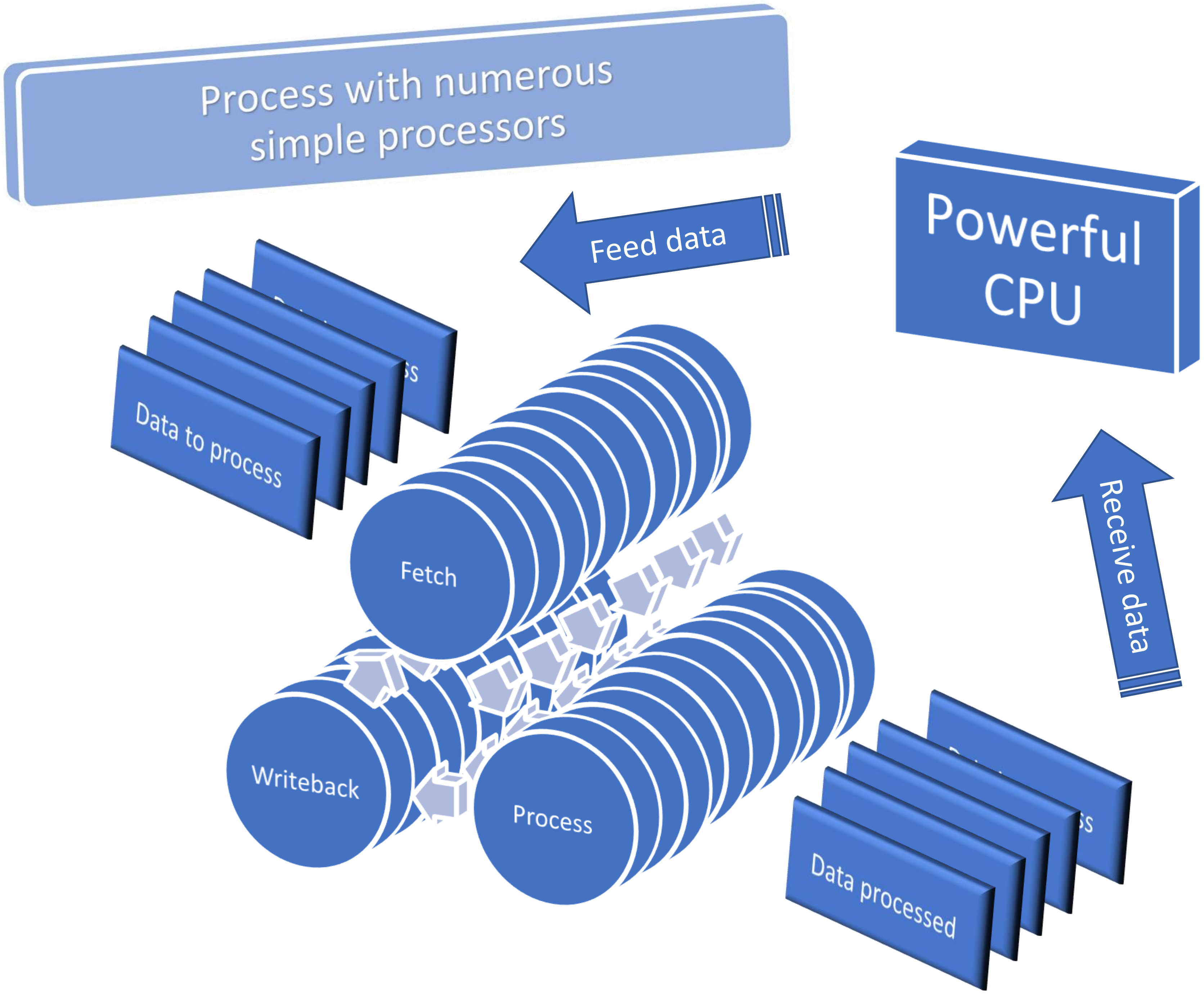
우주선 조

2014161001 강승우

KOREATECH Univ, ki6080@gmail.com

Process with single powerful CPU




Process with numerous simple processors
Powerful CPU
Feed data
Data to process
Fetch
Writeback
Process
Data processed
Receive data

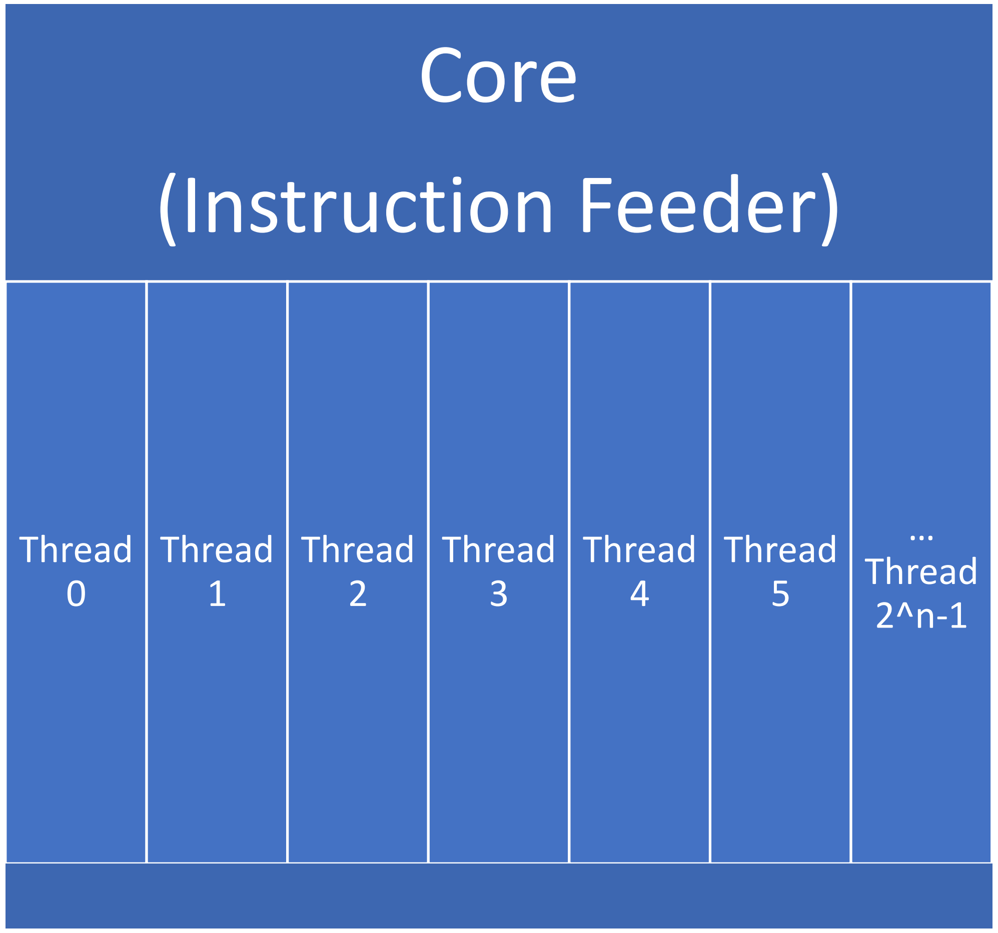
Concept

Single Instruction performs 2^n operations

Each thread has 32*32 bit general purpose registers.

No programmable instruction memory, no flow control on its own.

only logic & arithmetic operation supported.
Instructions should be fed from outside.



Goal

Advanced SIMD operation

- Numerous independent processors executes same command at the same time
- Fast local memory access

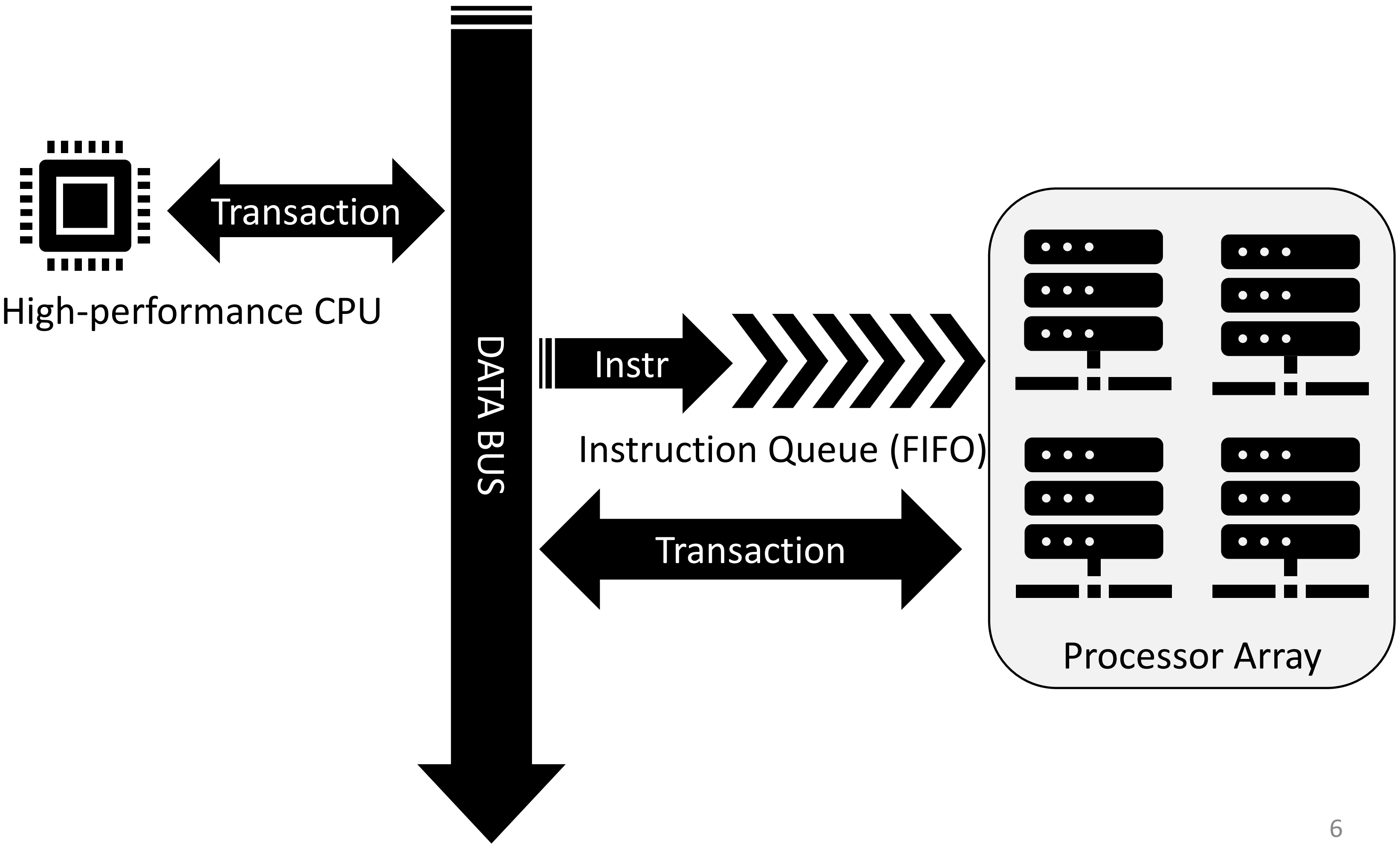
Platform Independence

- Synthesis level independence
  (IP Interfacing)
- Application level independence

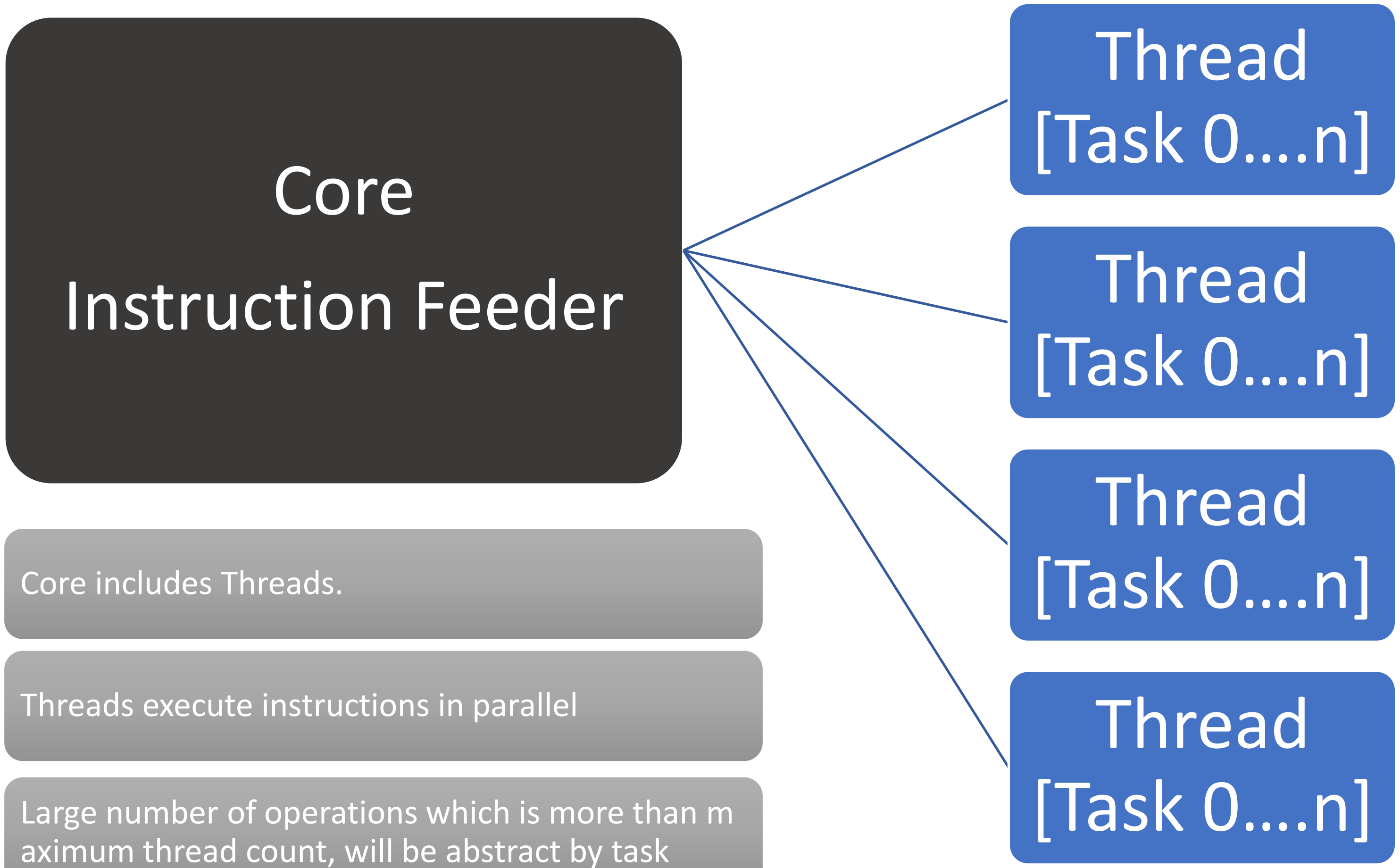
Ultra Fast Massive Data Processing

- 5 Stage pipelined architecture (Nested 2-stage pipelined FPU)
- Intensive RISC Instructions
- Can accept 200MHz input clock
  → 64 * 50MHz = 3.2GFlops per Core

System Overview



Core, Thread, Task

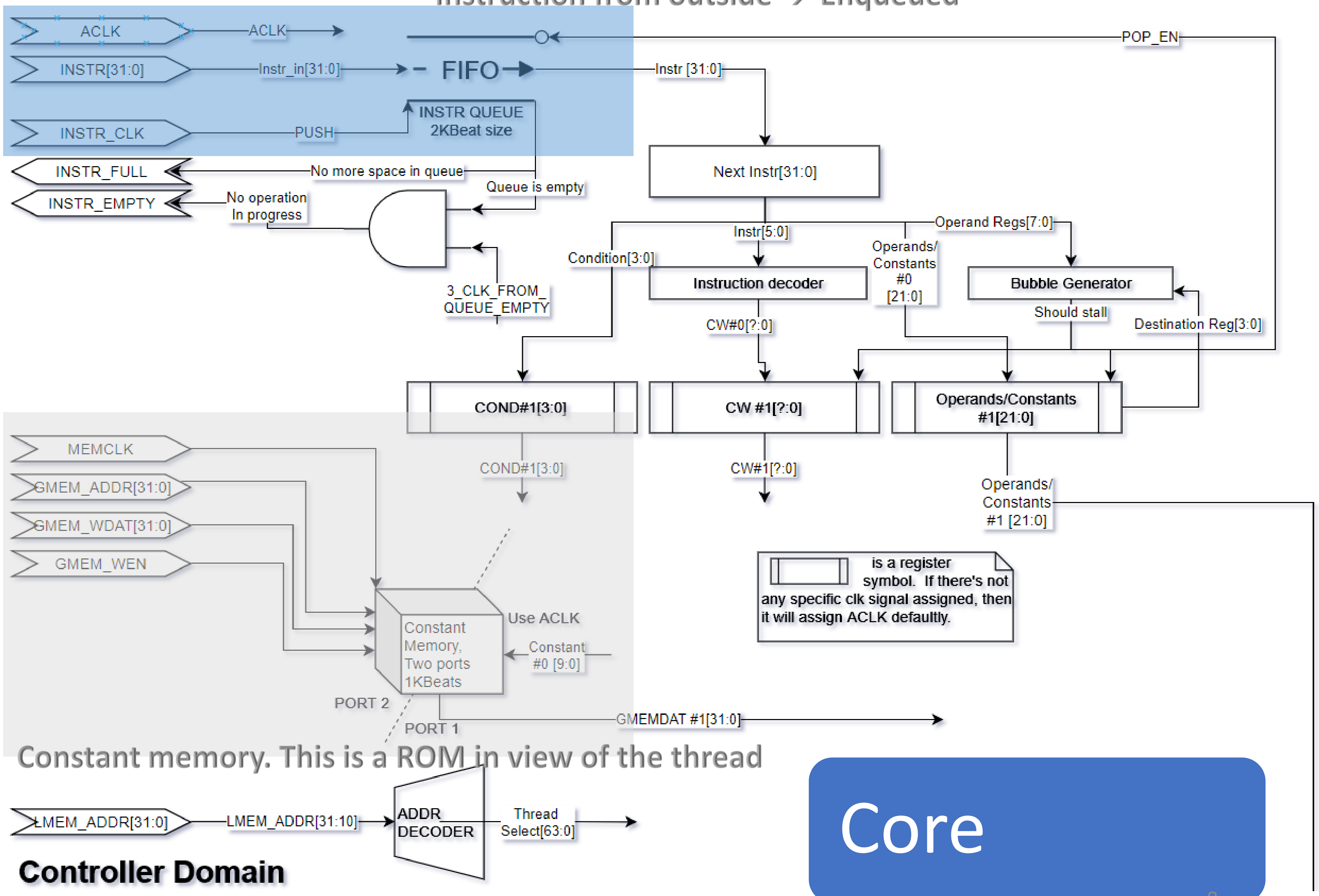


Core includes Threads.

Threads execute instructions in parallel

Large number of operations which is more than m aximum thread count, will be abstract by task

Instruction from outside → Enqueued

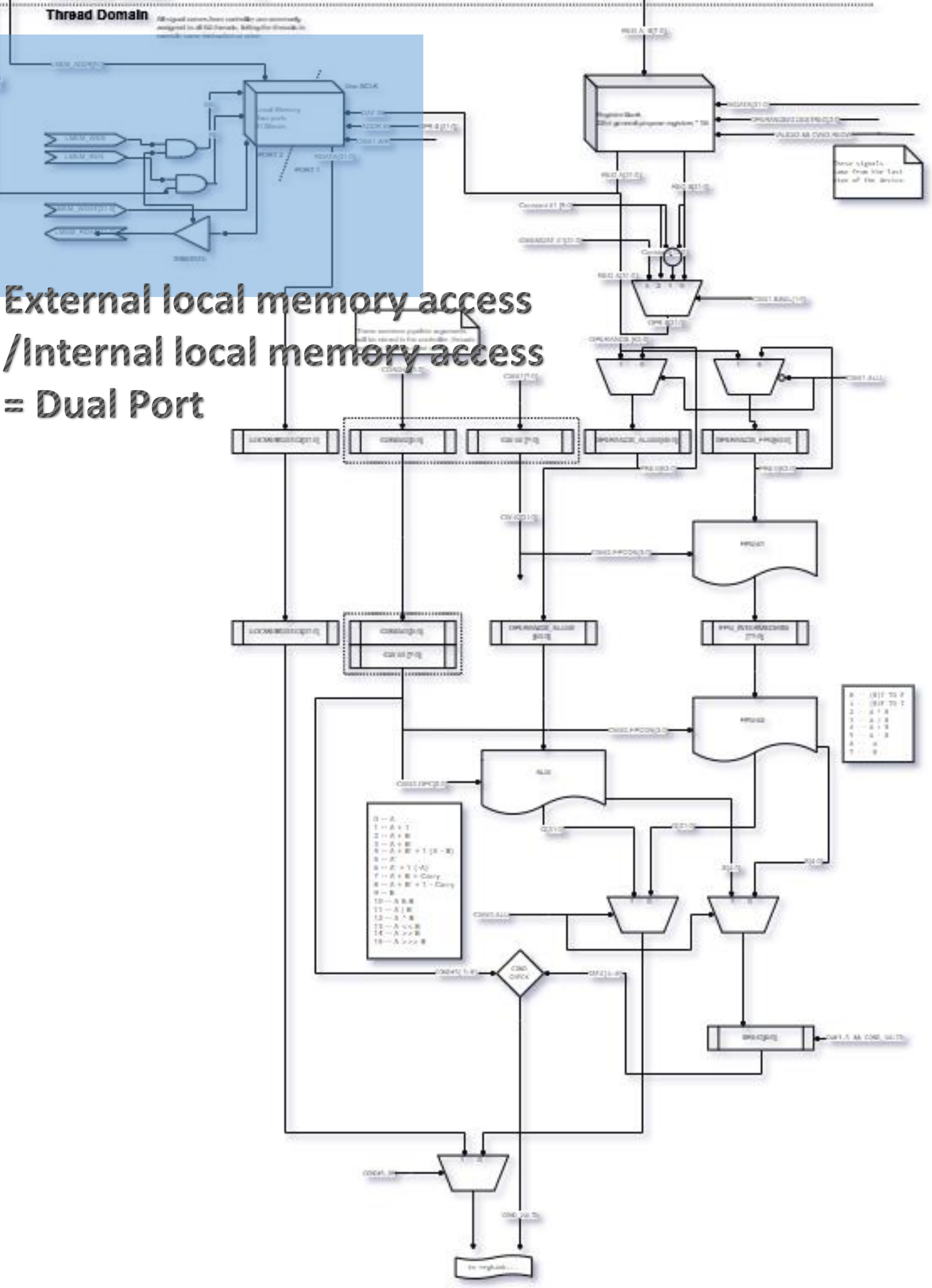


is a register symbol. If there's not any specific clk signal assigned, then it will assign ACLK defaultly.

Constant memory. This is a ROM in view of the thread

Controller Domain

Core

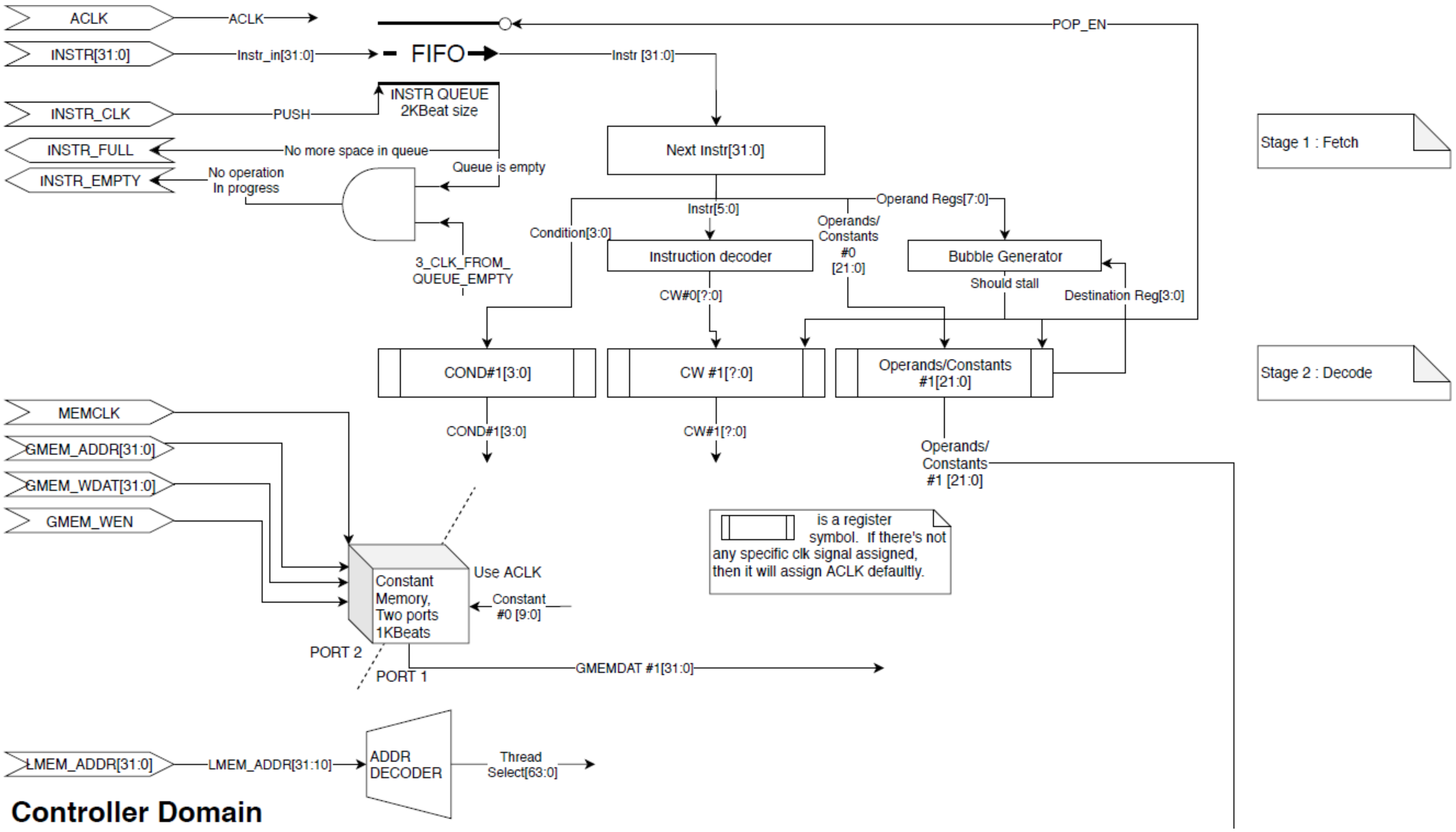


Thread

Pipelined FPU

Nested 1KBeat local memory (DP SRAM)

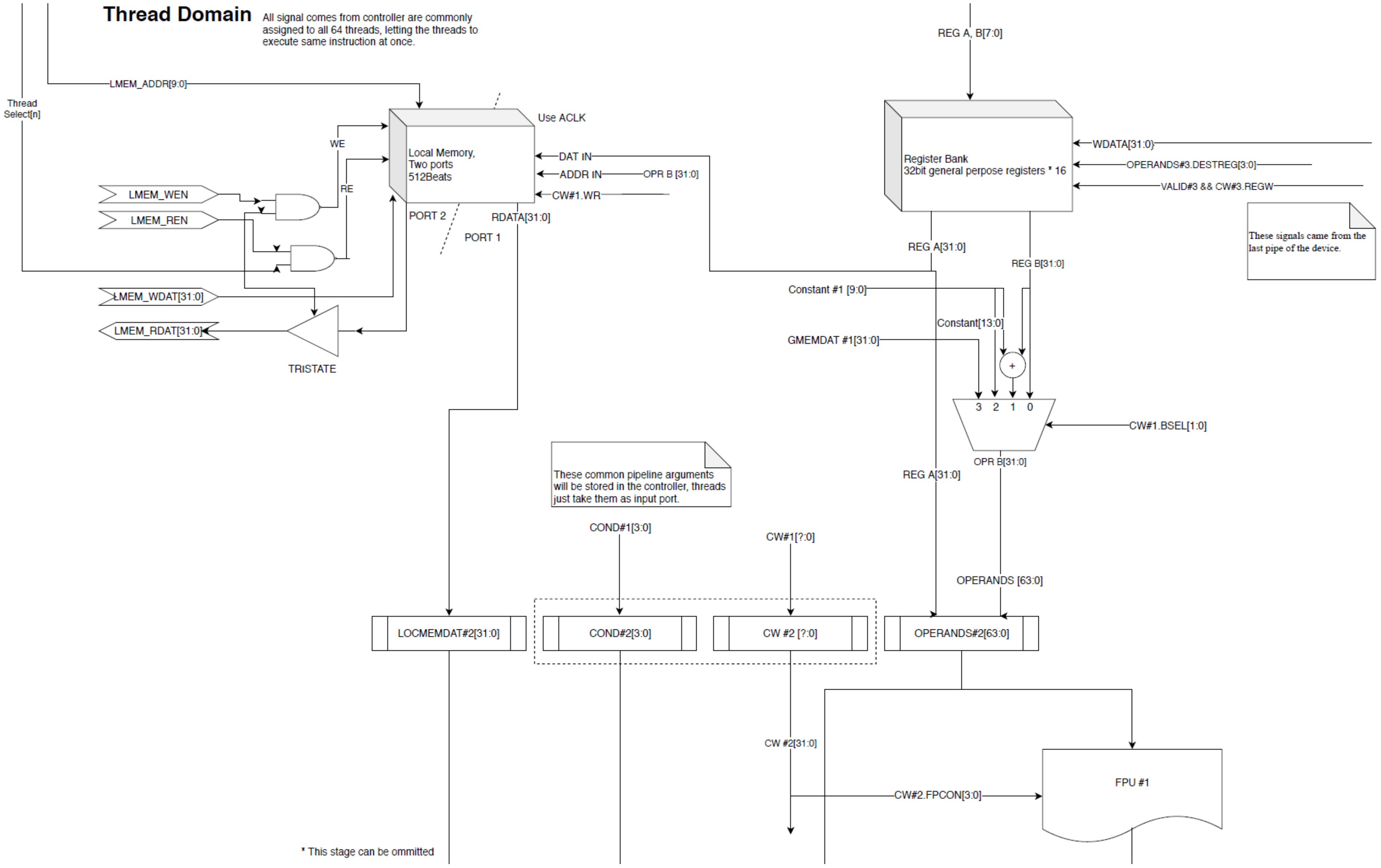
Conditional Instruction Execution

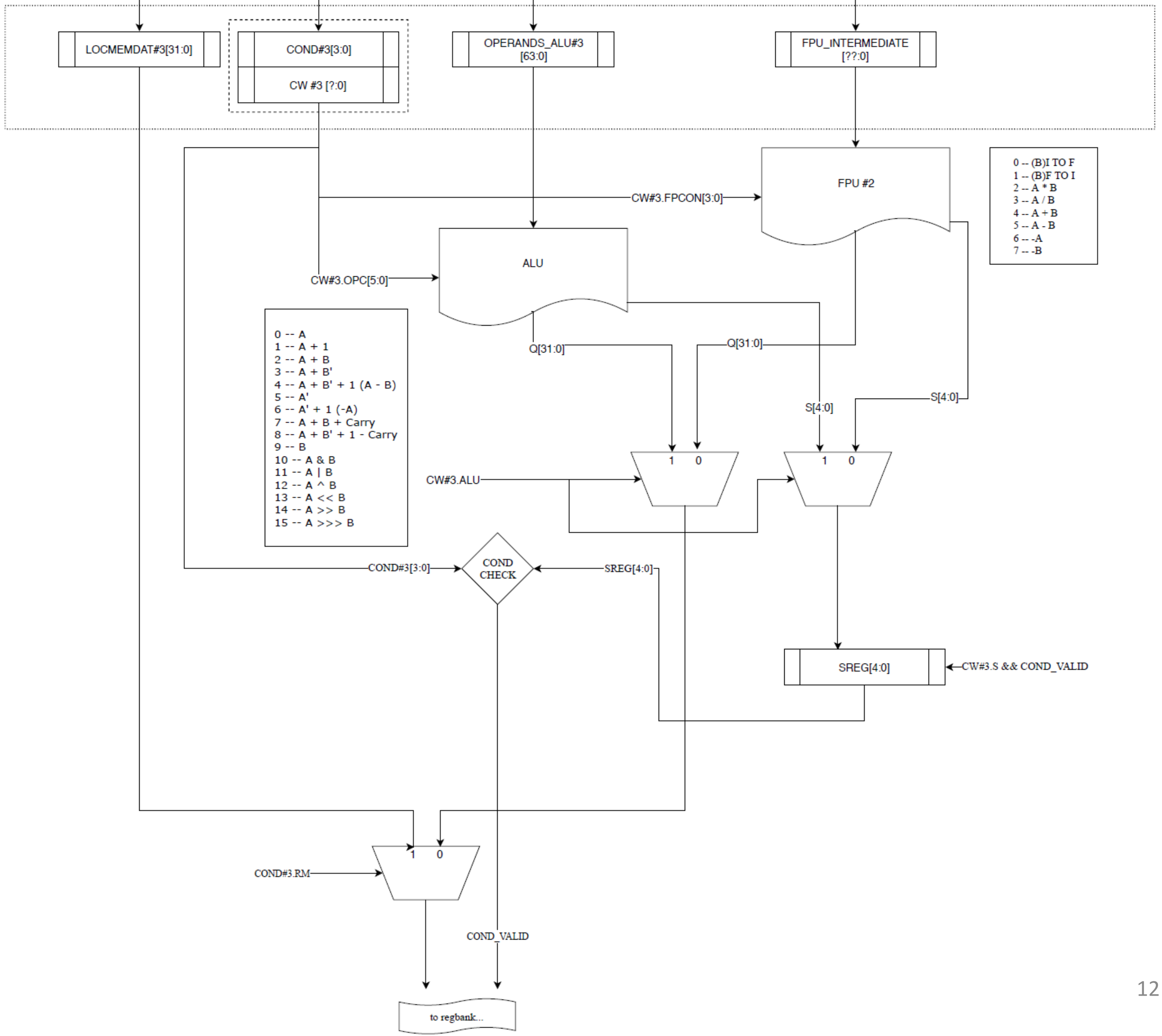


Controller Domain

Thread Domain

All signal comes from controller are commonly assigned to all 64 threads, letting the threads to execute same instruction at once.



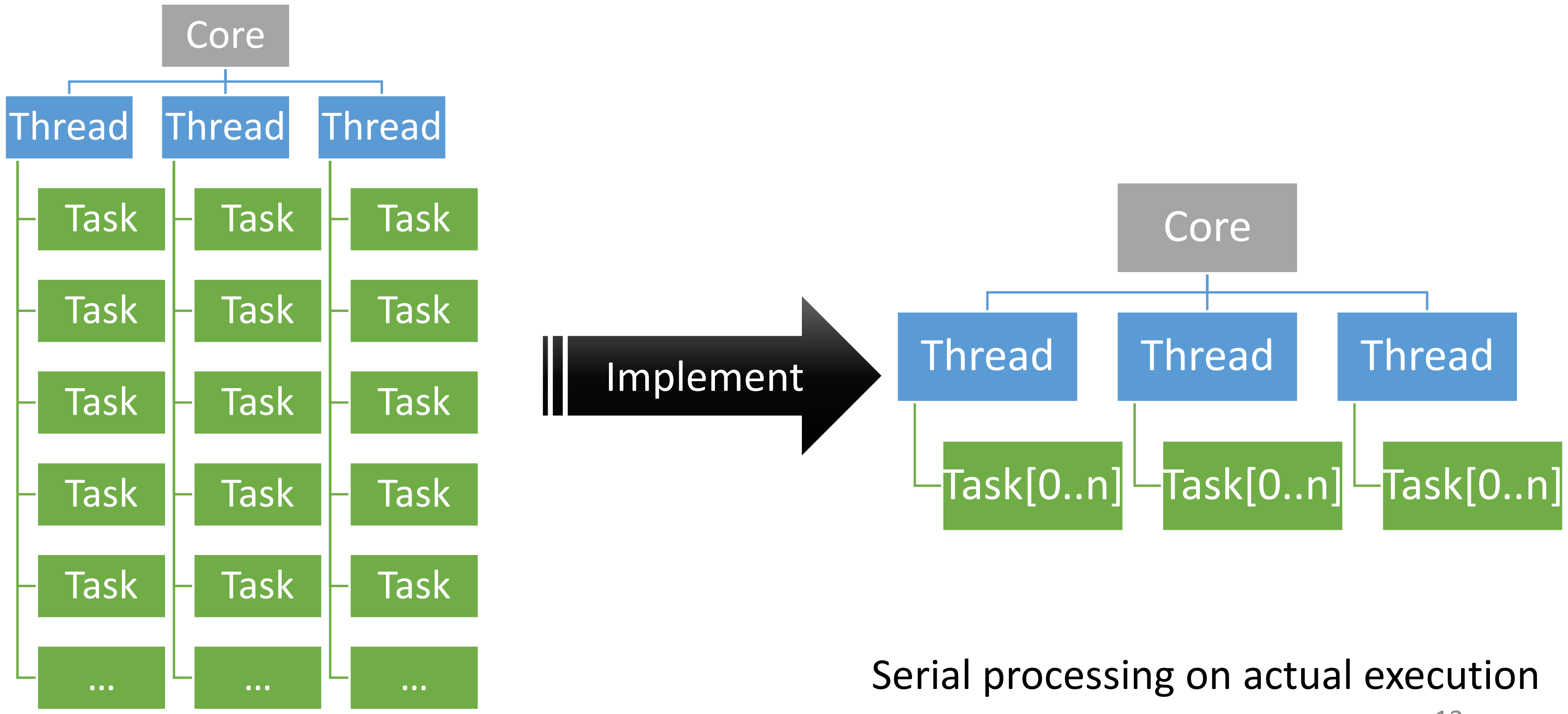


0	A
1	A + 1
2	A + B
3	A + B'
4	A + B' + 1 (A - B)
5	A'
6	A' + 1 (-A)
7	A + B + Carry
8	A + B' + 1 - Carry
9	B
10	A & B
11	A | B
12	A ^ B
13	A << B
14	A >> B
15	A >>> B

0	(B)I TO F
1	(B)F TO I
2	A * B
3	A / B
4	A + B
5	A - B
6	-A
7	-B

Task

Abstract parallel process



Serial processing on actual execution

Interface

- Data transaction
  - Global memory write (Constant ROM)
  - Local memory access
    - Write
    - Read
- Queue instruction
  - [Conditional] Arithmetic Operations
    - Floating point
    - Integer
  - Load/Store operation

Thread ISA

- 32 bit RISC Instruction
    - Morph of ARM Architecture
- 3 Instruction Models

COND	OPR	S	RD	RA	IMM	RB
31 28	27 23	22	21 17	16 12	11 5	4 0

COND	OPR	S	RD	RA	IMM
31 28	27 23	22	21 17	16 12	11 0

COND	OPR	S	RD	IMM
31 28	27 23	22	21 17	16 0

Thread ISA

ALU

- `MOV   D := M`
- `MVN   D := 'M+1`
- `ADC   D := A+M+C`
- `SBC   D := A+'M+'C`
- `AND   D := A&M`
- `ORR   D := A|M`
- `XOR   D := A^M`
- `ADI   D := A+I`
- `SBI   D := A-I`
- `MVI   D := I`

FPU

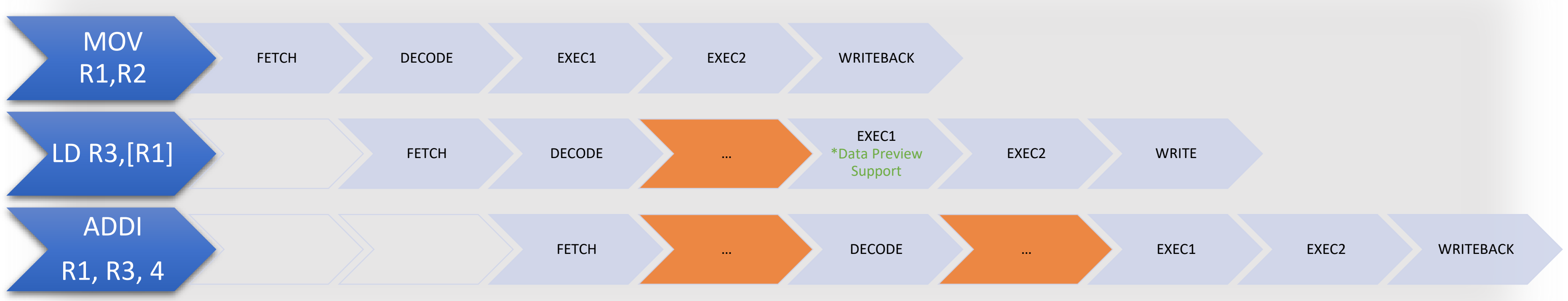
- ITOF
- FTOI
- FMUL
- FDIV
- FADD
- FSUB
- FNEG

GENERAL

- `LDL D := L[A+I]`
- `LDC D := G[A+I]`
- `LDCI D := G[I]`
- `STL L[B+I] := A`

- `L : Local memory`
- `G : Const memory`

Instruction Level Parallelism



- 5-stage pipeline
- Hardware prevents data hazard by inserting bubble
- CPI decreases on every bubble…

Instruction Level Parallelism

- Independent tasks run on single core at the same time.
- There are 32 Registers inside of each thread
- Simulate two internal parallel process,
  assigning 16 registers per task

```
mvi r15, #0
mov r0, r1
add r2, r0, r3
...
mvi r15, #16
mov r0, r1
add r2, r0, r3
```

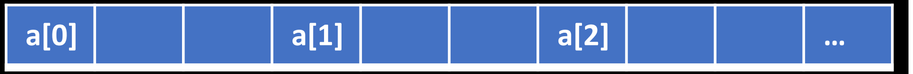
```
mvi r15, #0
mvi r31, #16
mov r0, r1
mov r16, r17
add r2, r0, r3
add r18, r16, r19
...
```

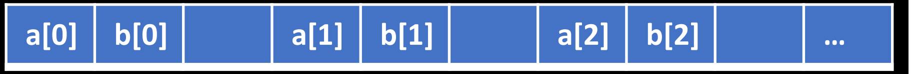
No Stalls Anymore!

This is not hardware-supported,
following cpp code will represent example of this paradigm

Code Example

```
void mult(float const* a, float const*b, size_t NumData, float* dst)
{
    CalcDevice pc;
    pc.ClearData();
    pc.SetSpacePerTask(sizeof(float)*3);
    pc.WritePerTask(
        /*Target data*/a,
        /*Number of tasks*/NumData,
        /*Ofst from task space*/0,
        /*Size per write*/sizeof(float));
    pc.WritePerTask(
        /*Target data*/b,
        /*Number of tasks*/NumData,
        /*Ofst from task space*/4,
        /*Size per write*/sizeof(float) );
```





Code Example

```
pc.ClearTaskQueue();
pc.QueueTaskLoadWord(
    // This operation will execute
    //  LDR DST, [r15 + OFST]
    // per task
    /*Target Register*/ r1,
    /*Data Ofst*/0);
pc.QueueTaskLoadWord(
    /*Target Register*/ r2,
    /*Data Ofst*/4);
pc.QueueTaskFMul(
    /*Target Register*/ r0,
    /*OPR A, B*/r1,
     r2);
pc.QueueTaskStoreWord(
    /*Target Register*/ r0,
    /*Data Ofst*/8);
```

a[0]	b[0]	a[0] *b[0]	a[1]	b[1]	a[1] *b[1]	a[2]	b[2]	a[2] *b[2]	…

Code Example

```
    pc.ExecuteTaskQueue();
    pc.Flush();
    pc.ReadPerTask(
        /*Destination*/dst,
        /*Number of tasks*/NumData,
        /*Ofst from task space*/8,
        /*Size per read*/sizeof(float)
    );
}
```

A kind of return procedure.

Milestone

Design architecture & ISA

Implement pipelined FPU

Implement GPPCU device

Composite overall system
(As Memory Mapped Device)

Program application (Device driver, Application)

Progress
On Week 5, May 2019

Concept

Week 3, May 2019 Thread Concept	Week 3, May 2019 Core Concept	Week 3, May 2019 Device Concept	Week 4, May 2019 Interface Concept	Week 4, May 2019 User-Level Usage Concept

Layout

Week 4, May 2019 Core Diagram	Week 5, May 2019 Thread Diagram	Week 5, May 2019 System Diagram	Week 1, June 2019 Thread ISA

Implementation

On Progress Pipelined FPU	Processor control word design	Pipeline design (Minimal prevent data hazard)

Thank you

Q & A