

Harmonica GPU

©Chad Kersey and Sudhakar Yalamanchili unless otherwise noted

(1)

Objectives

- Detailed look at the implementation of a SIMT GPU
- Example of the type of information propagated down the pipeline
- Basis for the next assignment and the default project

(2)

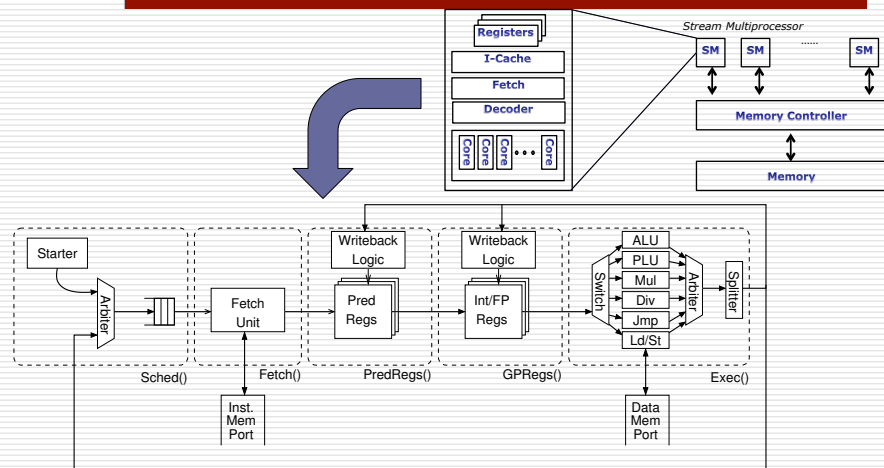
- C. Kersey, “HARP Instruction Set Manual”
- CHDL Architecture Description
 - ❖ <https://github.com/cdkersey/harmonica2>
 - ❖ Note that this is under active development so the current code base/definitions may be changing

(3)

- An instance of the HARP family of ISAs
 - ❖ SIMT oriented RISC-like ISAs
- Parametric SIMT GPU
 - ❖ Datapath width, instruction encodings, #registers are configurable
- Lightweight, simplified pipeline with multi-warp execution
 - ❖ No thread blocks
 - ❖ Nested parallelism
- Designed as a memory side or CPU accelerator for data intensive computing

(4)

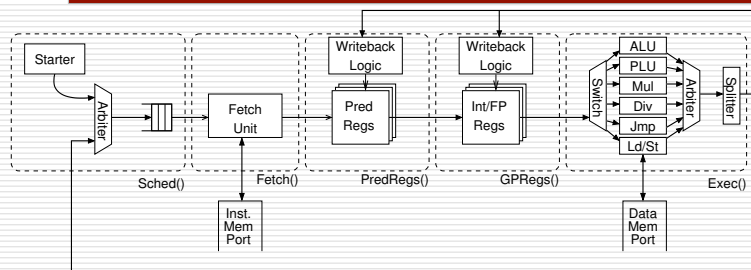
System Organization



- Parametric architecture
 - Key Parameters - #SMs, #lanes/SM, #registers, and data-widths.

(5)

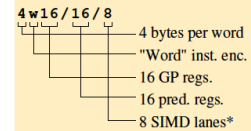
Microarchitecture



- Customizable, multithreaded, SIMT soft core
 - Generated from an architecture specification
- Supported by a generated HARP Tool assembler/linker/emulator
- Small - ~1500 lines of C++

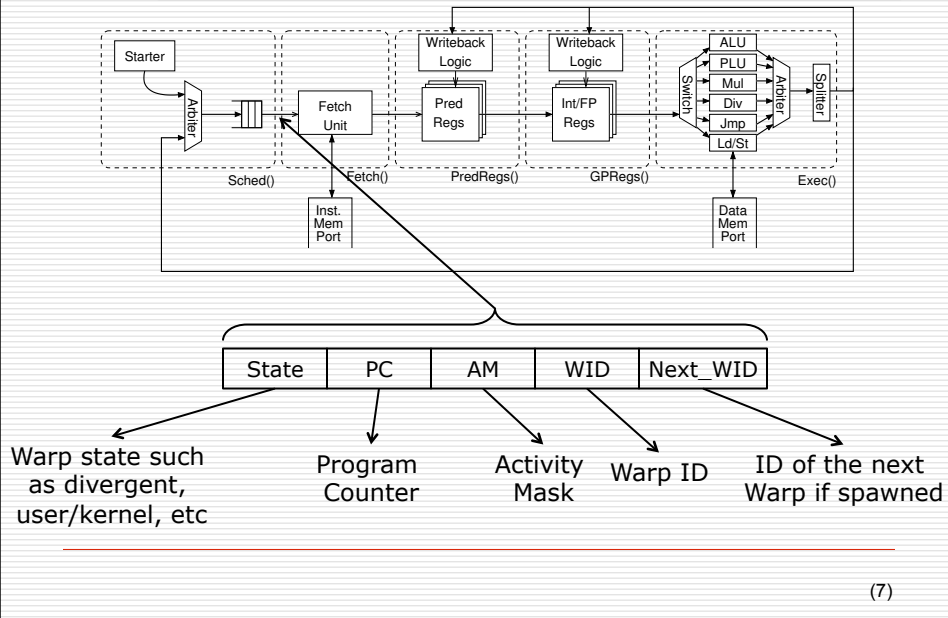
```
/* Return in r0 dot product of vectors
of real values pointed to by r0 and
r1, length in r2 */
dotprod: ldi r3, #0;
dploop: ld r4, [r0, #0];
ld r6, [r1, #0];
subi r2, r2, #1;
addi r0, r0, __WORD;
addi r1, r1, __WORD;
rtop @p0, r2;
fmul r4, r4, r6;
fadd r3, r3, r4;
@p0 ? jmp1 dploop;
ori r0, r3, #0;
jmpl r5;
```

ArchID String

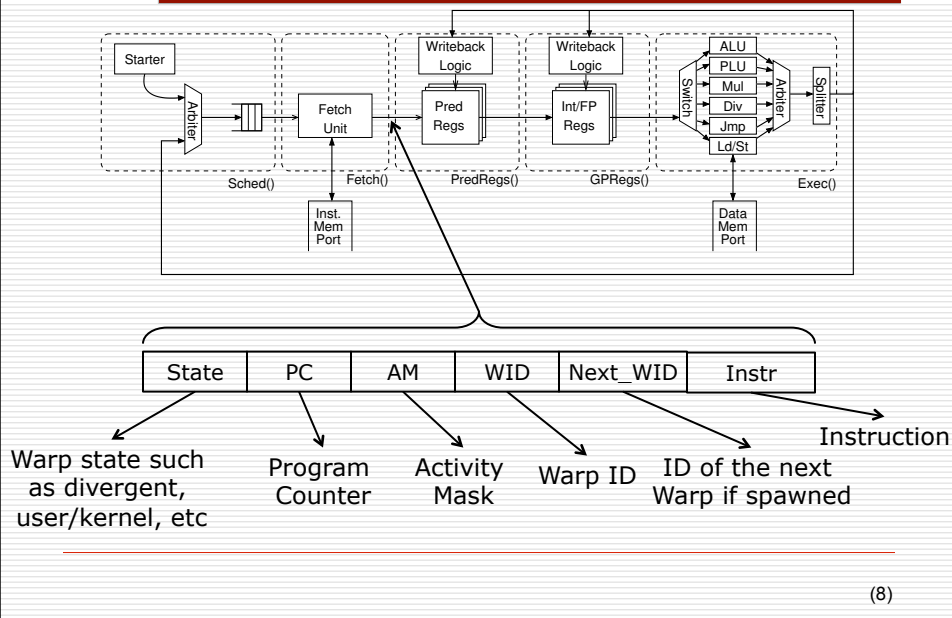


(6)

Microarchitecture: Schedule

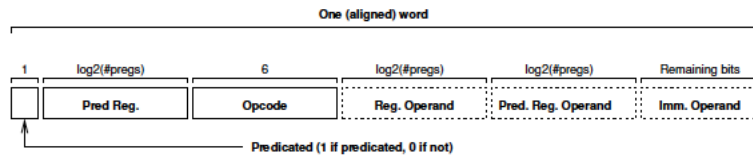


Microarchitecture: Fetch

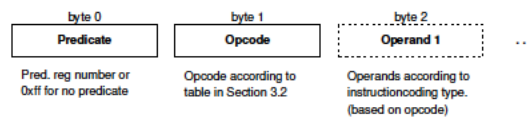


Harmonica Instruction Format

Word Encoding

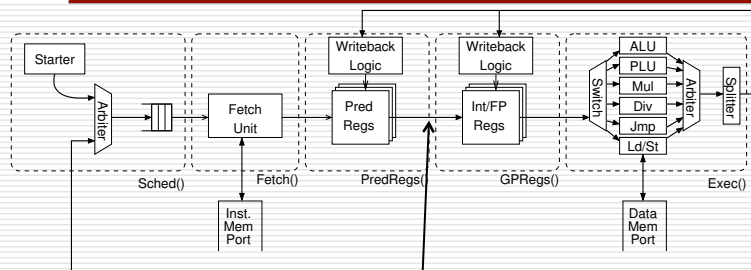


Byte Encoding

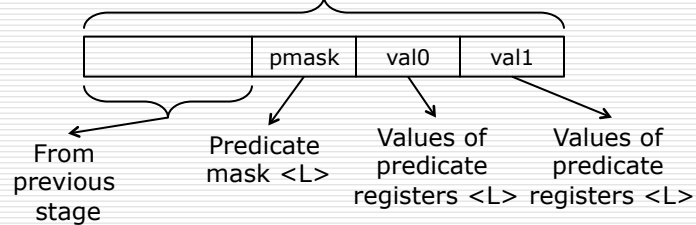


(9)

Microarchitecture: PRF Access

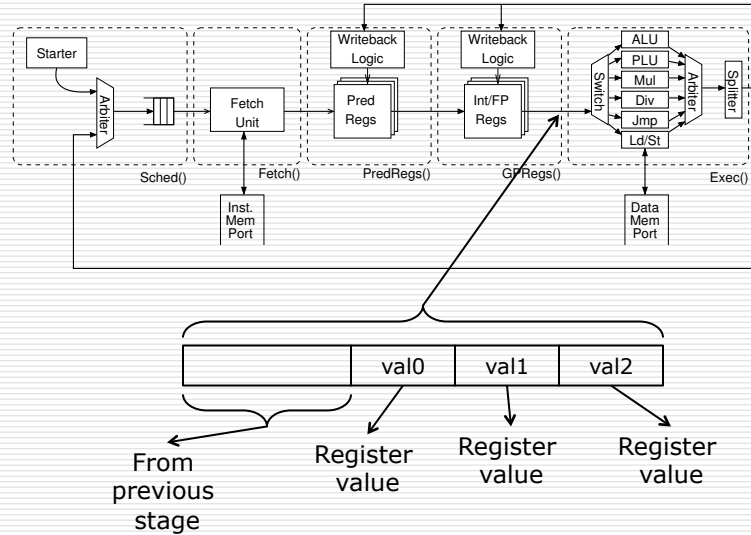


$L = \#lanes$



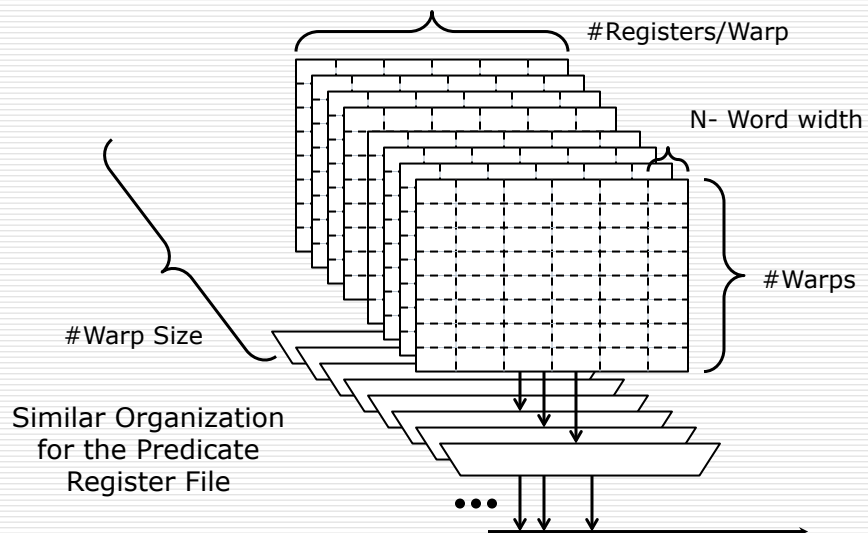
(10)

Microarchitecture: GPR Access



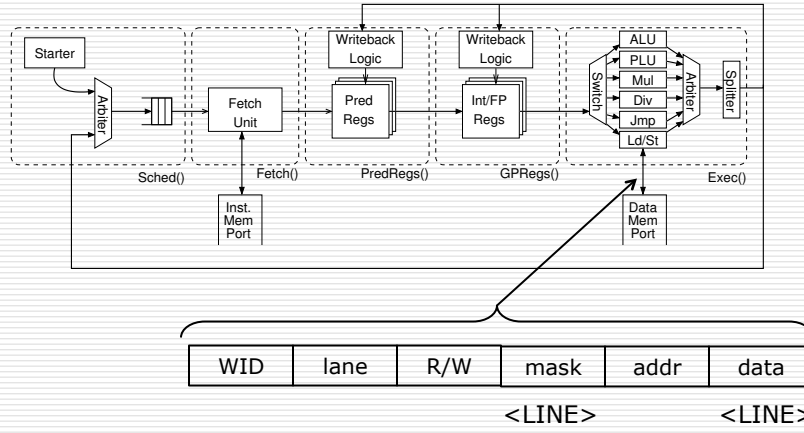
(11)

Register File Organization



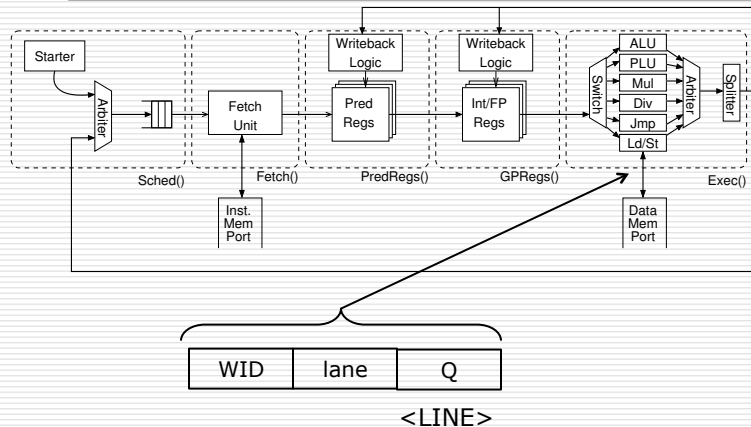
(12)

Microarchitecture: Cache Request



(13)

Microarchitecture: Cache Response



- Q – cache line number of words

(14)



(15)

The diagram illustrates the high-level architecture of the RISC-V processor, showing the flow of data and control signals through various stages and units.

Stages and Units:

- Starter:** Initiates the process.
- Arbiter:** Manages the flow of instructions.
- Fetch Unit:** Retrieves instructions from the **Inst. Mem Port** (Instruction Memory Port).
- Pred Regs (Predicates Registers):** Stores predicate results.
- Int/FP Regs (Integer/Float Registers):** Stores integer and floating-point data.
- ALU (Arithmetic Logic Unit):** Performs arithmetic and logical operations, including **PLU** (Pseudo-Logical), **Mul** (Multiply), **Div** (Divide), **Jmp** (Jump), and **Ld/St** (Load/Store).
- Splitter:** Distributes results to different execution paths.

Control and Data Flow:

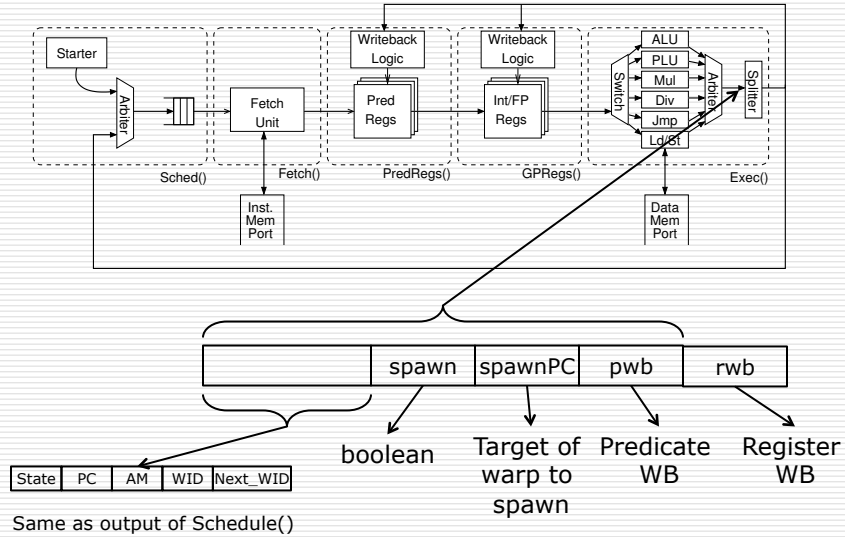
- Sched():** Scheduling function.
- Fetch():** Fetching function.
- PredRegs():** Predicate registers output.
- GPRs():** General Purpose Registers output.
- Exec():** Execution output.
- Data Mem Port:** Data Memory Port.

Instruction Format:

The instruction format is shown as a sequence of fields: **data**, **id**, **R/W**, **llsc**, and **llsc_suc**. The **id** field is labeled as **Unique ID to pair rq & rply**. The **llsc** field is labeled as **Load link store conditional**. The **llsc_suc** field is labeled as **Success**.

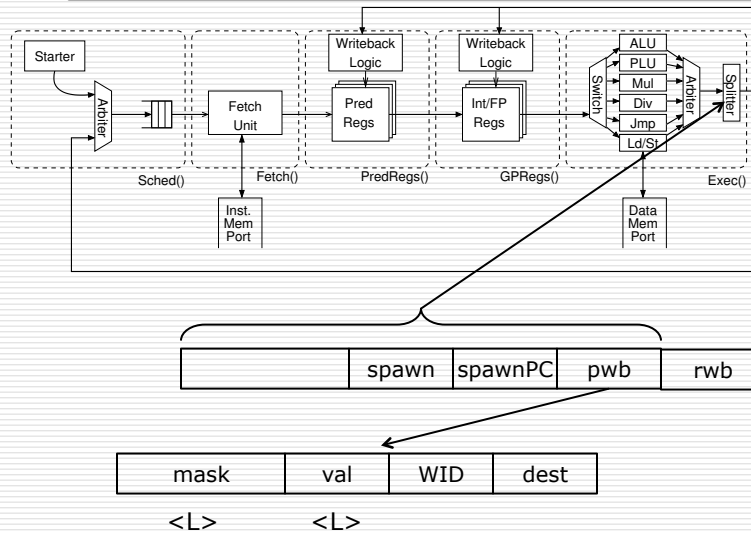
(16)

Microarchitecture: FU Out



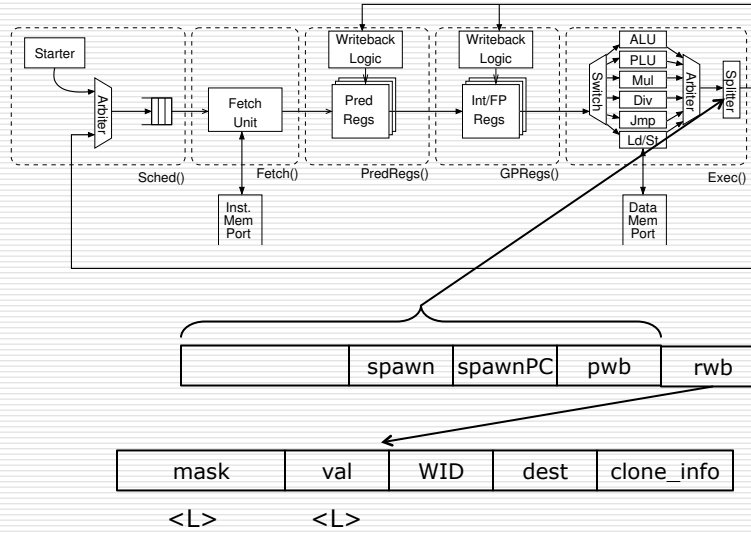
(17)

Microarchitecture: FU Out (2)



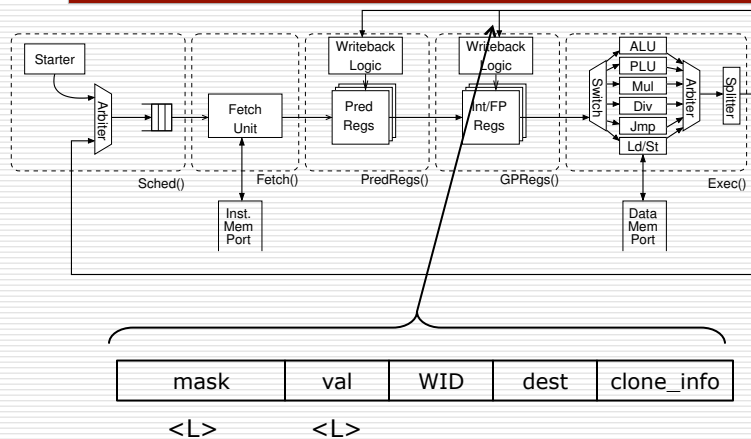
(18)

Microarchitecture: FU Out (3)



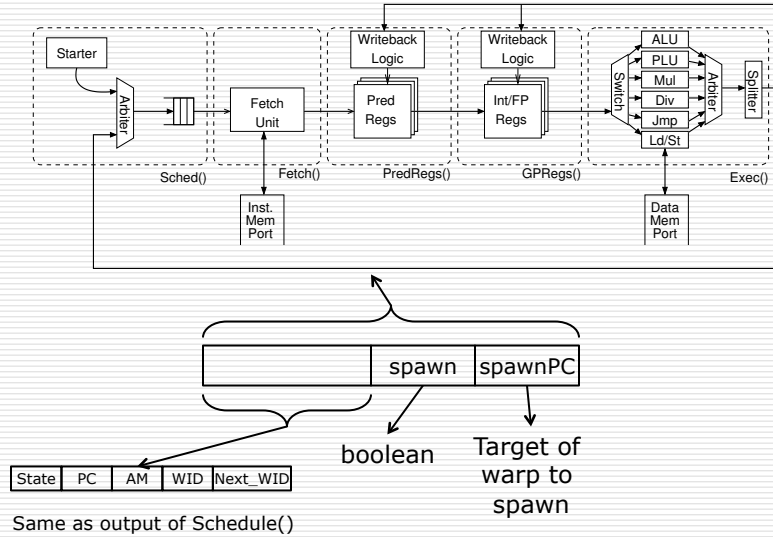
(19)

Microarchitecture: GPR Writeback



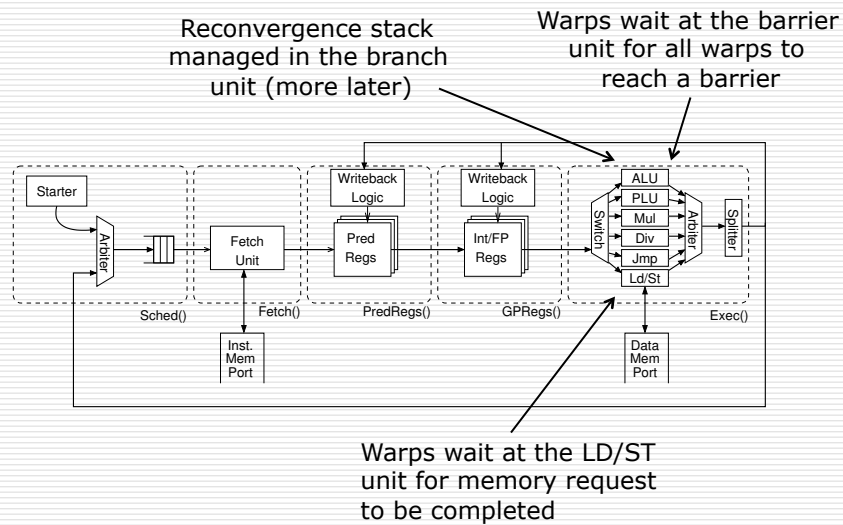
(20)

Microarchitecture: Next Instruction



(21)

Microarchitecture: Stalled Warps



(22)

Some Miscellaneous Observations

- Recirculating warp model of execution
 - ❖ Waiting warps distributed through the data path
 - Barriers, divergent warps, memory accesses
- Warps can spawn other warps → nested parallelism
- Elementary scheduling, and fetch policies
- Supports kernel mode execution and interrupts (lane 0)