

# FirePath™ Processor Architecture and Microarchitecture

Sophie Wilson

Chief Architect, Broadcom DSL BU

Rich Porter

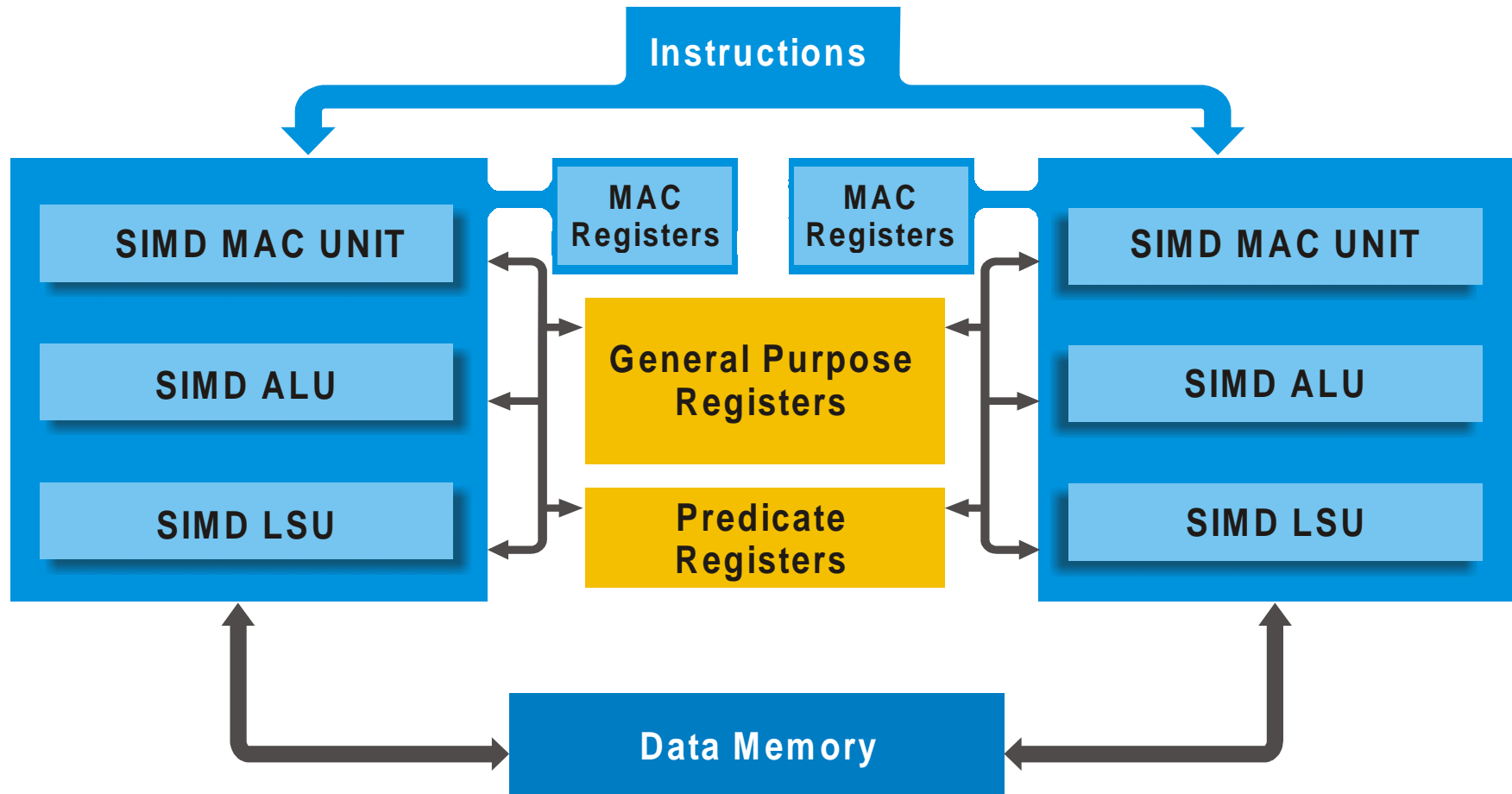
Staff Design Engineer, Broadcom DSL BU

# FirePath History

---

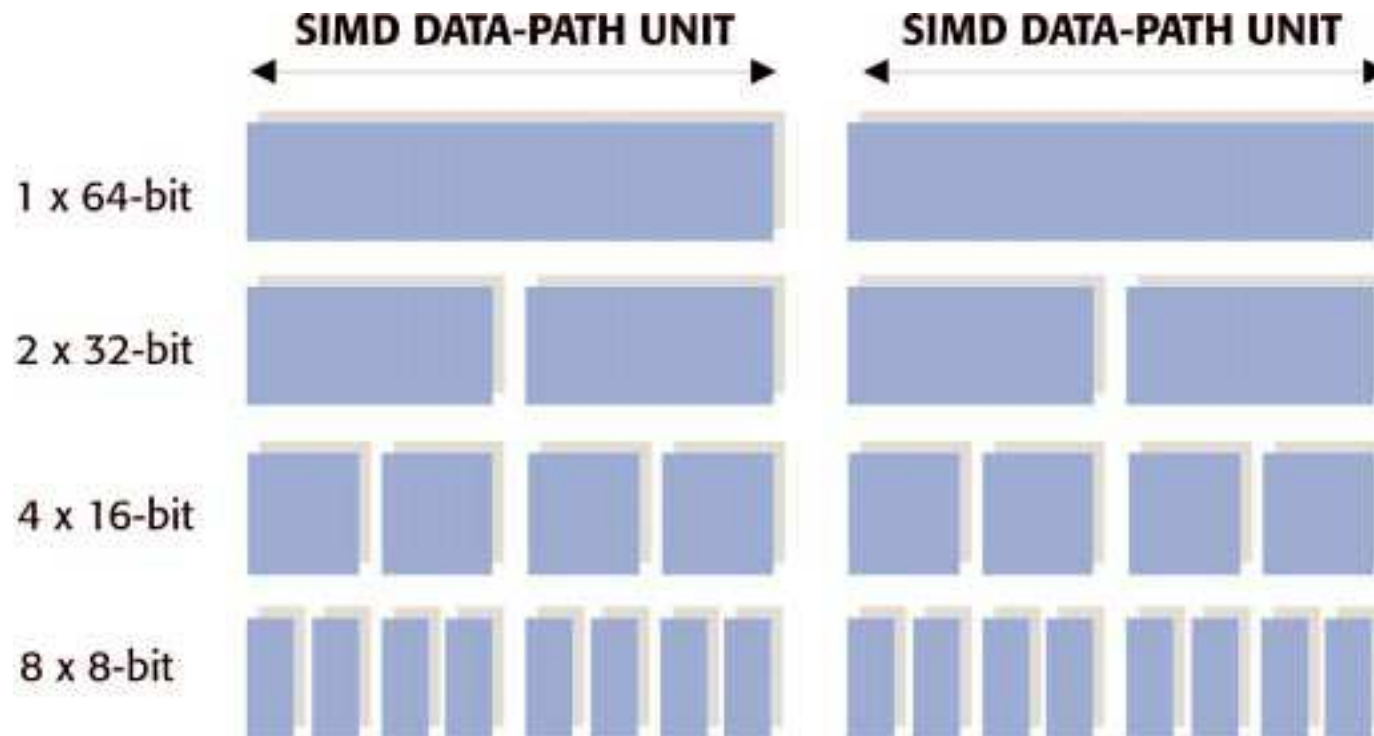
- FirePath - new processor designed by Element 14
- Element 14 formed 26<sup>th</sup> July 1999
- Element 14 acquired by Broadcom Nov 2000
- FirePath targetted at “Bulk Data Processing”
- First FirePath SoC is the BCM6410, a single chip 12-channel DSL digital transceiver

# FirePath Processor



# FirePath Parallelism

---



- Two forms of parallelism:
  - LIW: two identical 64-bit data-paths (compiler friendly)
  - SIMD: each data-path is SIMD-laned

# FirePath Architecture: LIW

---

- 128-bit execution width via LIW/SIMD machine
  - Instruction level parallelism exploited via two symmetric 64-bit RISC pipelines with 64 common 64 bit registers
  - Data parallelism exploited via 2, 4 or 8-way SIMD within each RISC pipeline

# FirePath Architecture: ISA

---

- Instruction set
  - Complete and largely orthogonal SIMD set
  - Supports DSP and control code
  - Specific support for communications algorithms such as Galois field arithmetic

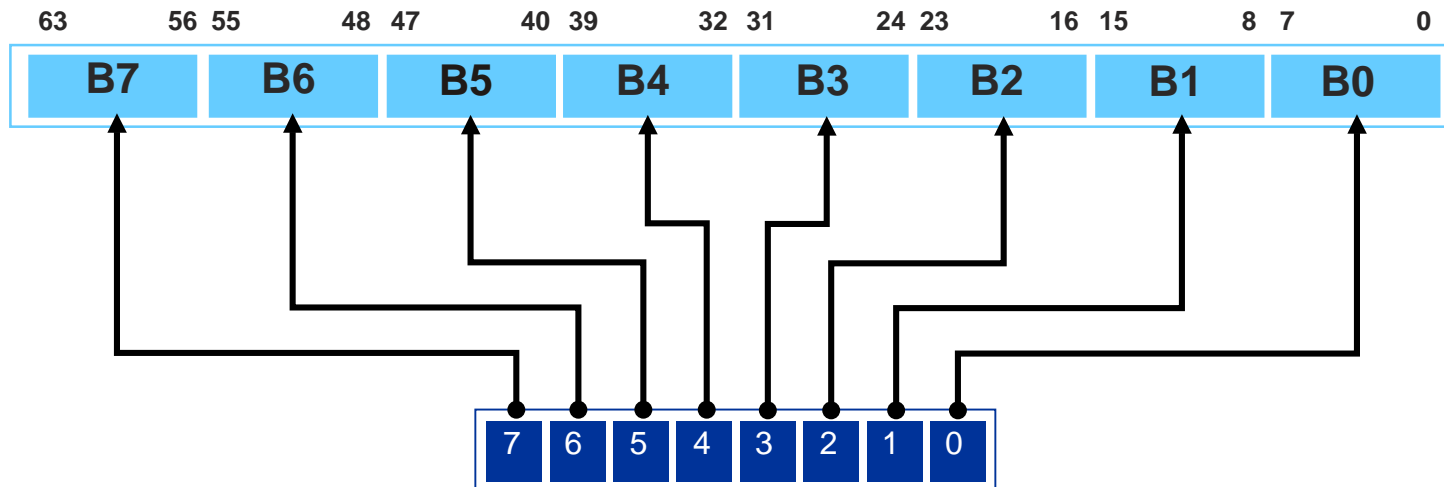
# FirePath Architecture: Compute

---

- Accessible compute power
  - Large general purpose register file (64 of 64-bit registers)
  - Integrated multiply-accumulate (MAC) unit with dedicated accumulator file
  - Per clock tick, can perform and sustain for example:
    - 8x16-bit load operations (load 128 bits),
    - 8x16-bit MAC operations,
    - 1 address pointer update

# SIMD Predication

- Predicate bit per byte lane:



8 bit predicate register

# What's not in FirePath Architecture?

---

- Precise exceptions, precise interrupts
- “DSP” address modes
- “DSP” zero-overhead branches
- Fixed binary coding
- Big-endian support

# FirePath Instruction Overview

Loads:	LD{S}B	LD{S}H	LDW	LDL	LDL1	LDL2
Stores:	STB	STH	STW	STL	STL1	STL2
Branch:	SBPF	SBPFL	SBPT	SBPTL		
Arith:	ADDB	ADDH	ADDW	ADDWB	ADDWT	
	CMP<>B	CMP<>H	CMP<>W			
	SUBB	SUBH	SUBW	SUBWB	SUBWT	
Logic:	ANDB	ANDH	ANDW	ANDL		
	BICB	BICH	BICW	BICL		
	EORB	EORH	EORW	EORL		
	ORRB	ORRH	ORRW	ORRL		
	TST<>B	TST<>H	TST<>W	TST<>L		
Misc:	CNTB	DEALB	EVENH	SUMG	CLZW	FFSW
Shift:	ASLH	ASRH	ASLW	ASRW	ASLL	ASRL
	LSLH	LSRH	LSLW	LSRW	LSLL	LSRL
		RORH	RORW	RORL		
Multiply- Acc:	MACH	MDCH	MNCH	MZCH		
	MACW	MDCW	MNCW	MZCW		
	MULG	MACG	MULH	MULW		

# Code example: FIR inner loop

;Perform MACs with maximal software pipelining - use old and load new at once

;Each stage moves input data on by a half word: 01234567 becomes 12345678

fir\_inner\_loop:

;MAC2s do 8 half word multiplies by first four coefficients in r7 half words 0..3

MAC2SSH m0/m1, s01234567, r7.h0 : LDL2 s01234567, [input, #2]

MAC2SSH m0/m1, s12345678, r7.h1 : LDL2 s12345678, [input, #4]

MAC2SSH m0/m1, s23456789, r7.h2 : LDL2 s23456789, [input, #6]

MAC2SSH m0/m1, s3456789a, r7.h3 : LDL2 s3456789a, [input, #8]

;decrement loop counter, and load next set of r7 coefficients (all used by now)

SUBWBS inner\_c, inner\_c, #1 : LDL r7, [coeff\_ptr, #8]

;MAC2s do 8 half word multiplies by first four coefficients in r7 half words 0..3

MAC2SSH m0/m1, s456789ab, r9.h0 : LDL2 s456789ab, [input, #10]

MAC2SSH m0/m1, s56789abc, r9.h1 : LDL2 s56789abc, [input, #12]

MAC2SSH m0/m1, s6789abcd, r9.h2 : LDL2 s6789abcd, [input, #14]

MAC2SSH m0/m1, s789abcde, r9.h3 : LDL2 s789abcde, [input, #16]!

;Test loop condition, and load next set of r9 coefficients (all used by now)

SBPFL p6.0, fir\_inner\_loop : LDL r9, [coeff\_ptr, #16]!

# FirePath™ Processor Micro Architecture

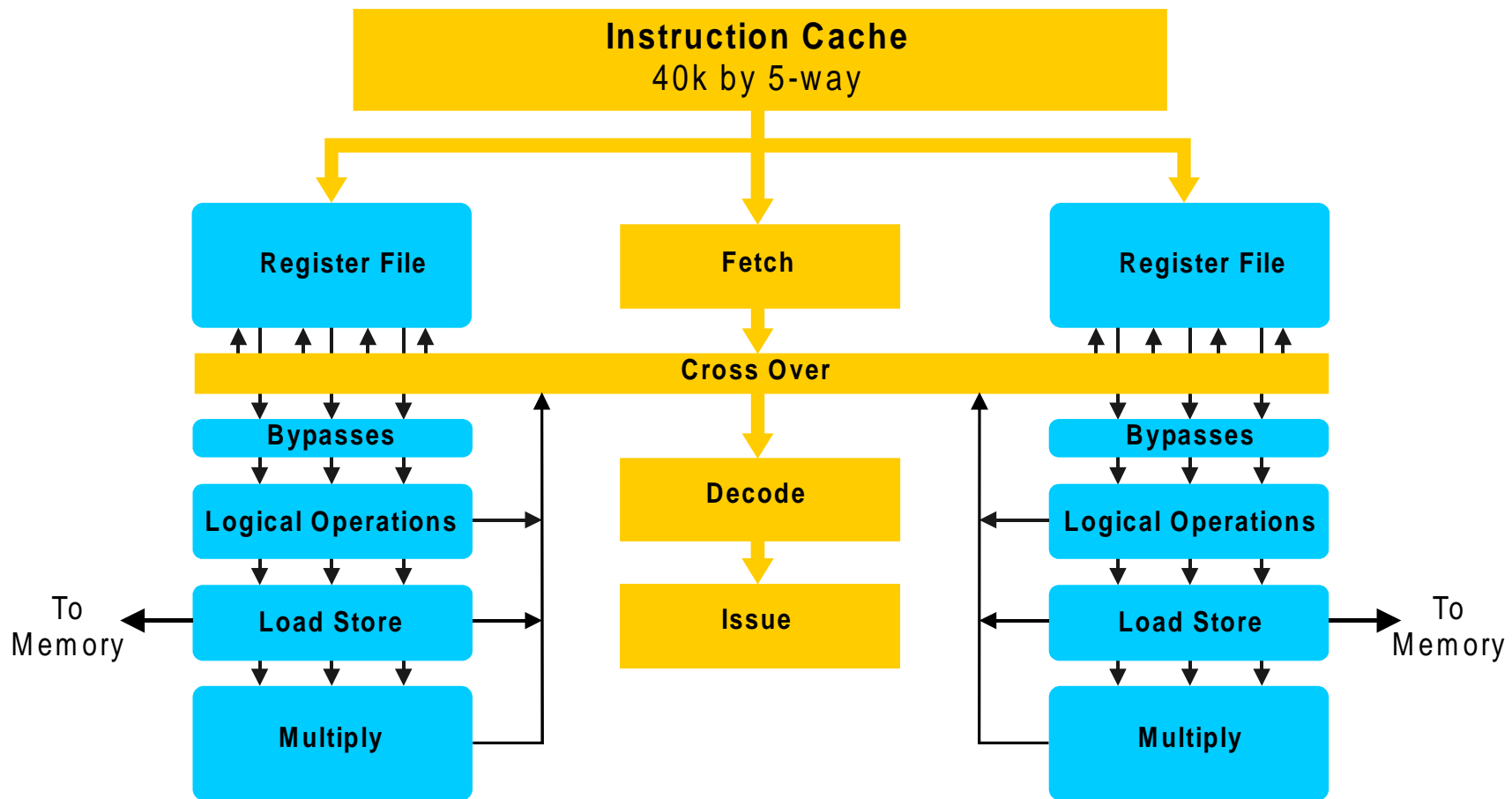
Rich Porter  
Staff Design Engineer,  
Broadcom DSL BU

# FirePath Introduction

---

- Instruction stream 64 bits
  - In order issue
  - Out of order completion
- SIMD predicated execution
  - Single cycle arithmetic and logic
  - 6 Cycle load store
  - 7 Cycle multiply capable of back to back multiply accumulate instructions

# Micro Architecture Schematic



# Register File

---

- Instructions have high register bandwidth
  - Each side may read three and write two general purpose registers

p0.DEALB r0/r1, r2, r3/r4 : p0.DEALB r10/r11, r12, r13/r14
  - x side {src r2,r3,r4 dest r0/r1}, y side {src r12,r13,r14 dest r10/r11}
  - Instruction also implies resource availability

MUL2W r0/r1, r2/r3, r4/r5 : LDL2 r6/r7, [r8]
  - x side {lsu{src r8, dest r7}, mul\_pipe{src r2, r4, dest r0}}
  - y side {lsu{src r8, dest r6}, mul\_pipe{src r3, r5, dest r1}}
- Requirement of six read and four write ports
  - Ten port register file represents design challenge
- Write ports also have byte enable

# Register Files

---

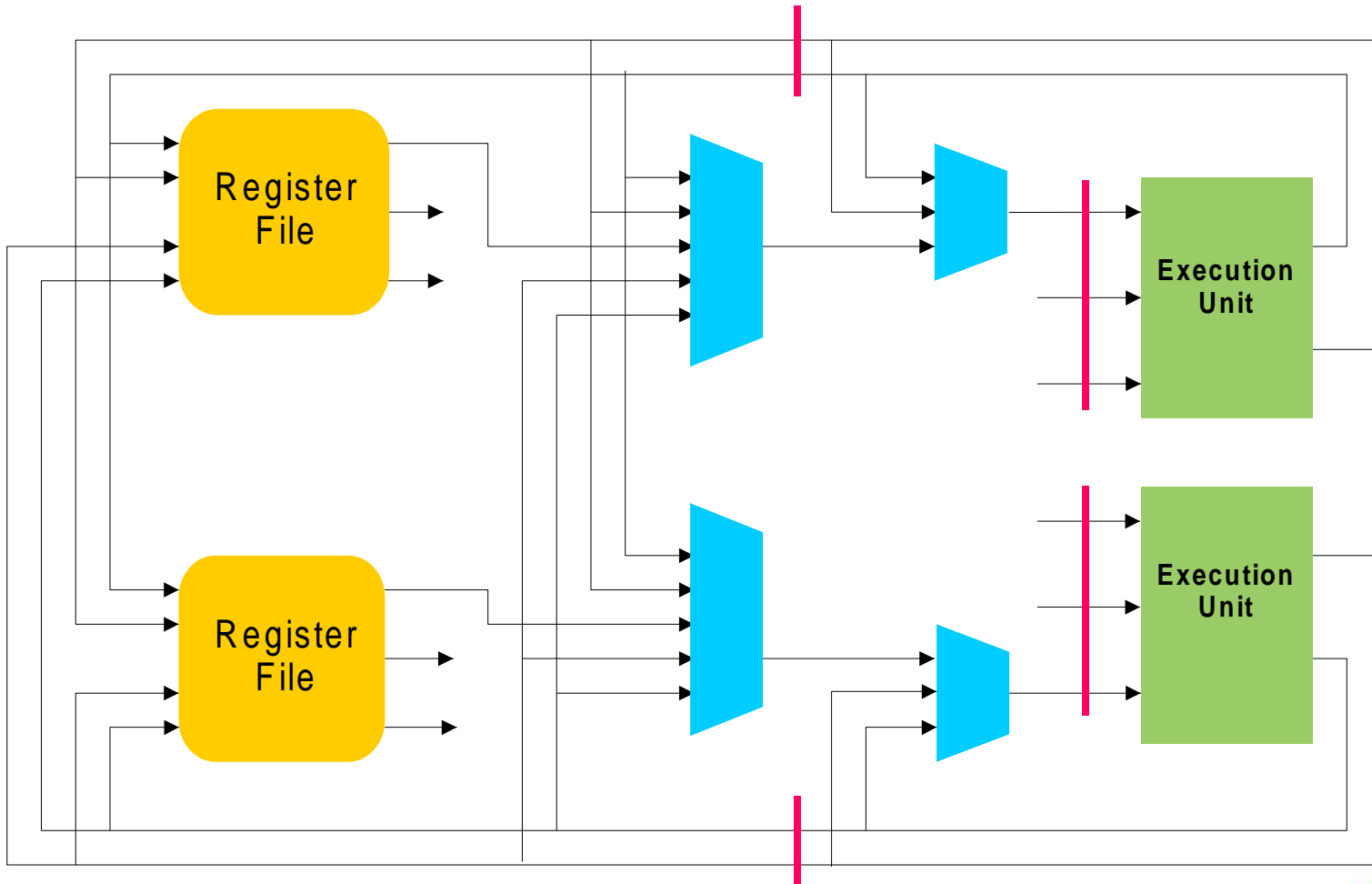
- Reduce wiring by duplication
  - Two datapaths, two register files
  - Pushing some timing problems back to WB stage
- Read ports perform some decode
  - Instructions that require 'pair' of values decoded by register file which presents both values
  - Program Status Register determines which high order registers to use
- Early presentation of WB addresses allows drop through
  - Reducing bypassing always a good thing

# Bypassing

---

- FirePath is almost fully bypassed
  - Single cycle instructions
  - zero penalty same side
    - DEALB r0/r1, r2, r3/r4 : something
    - DEALB r8/r9, r2, r0/r1 : something else
  - single cycle penalty other side
    - AND r0, r1, r2 : something
    - something else : AND r3,r0,r4 ; stop for 1 cycle  
; (in order issue)
  - Multiple cycle instructions
  - value available as soon as instruction completes (same side)
  - plus one cycle other side

# Bypassing Schematic



# Hierarchical Bypassing

- Predication causes complication

- A register value can exist in parts in different places

p0.AND r0, r1, r2

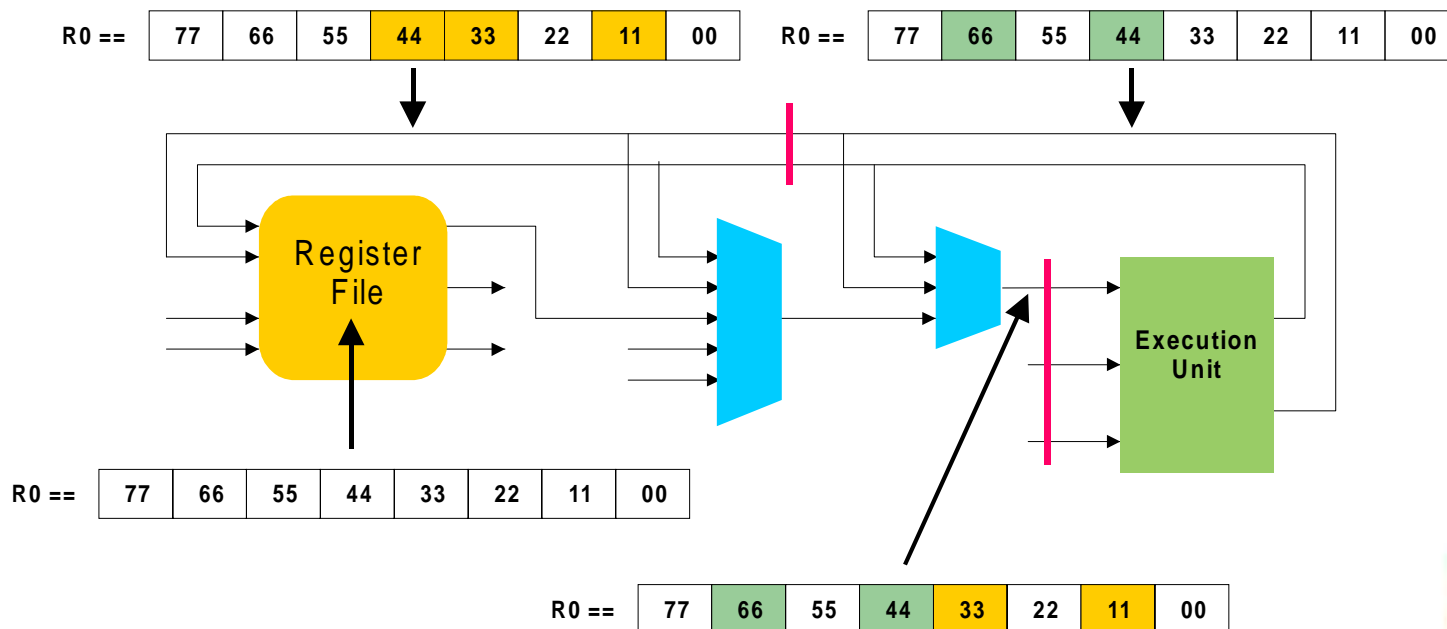
; p0 == 0x1a

p1.AND r0, r3, r4

; p1 == 0x50

ORR r5,r0,r6

; r0 in {register file, WB, EX1}



# Multiply Pipeline

---

- Each side has seven stage pipe
  - Capable of two 32 bit or four 16 bit multiplies per instruction
  - Can issue instructions back to back
- Two registers for multiply accumulate instructions
  - 40 bit for 16 bit data, 64 bit for 32 bit data
  - Distinct per side, no hazards
- Shifting and rounding stages
- Pipelines are independent
  - In order issue, out of order completion

# Multiply Pipeline more

---

- 16 bit

MULH	MUL0	MUL1		ACC0	SHIFT0	SHIFT1	SHIFT3	WB
------	------	------	--	------	--------	--------	--------	----

MACH	MUL0	MUL1		WB				
------	------	------	--	----	--	--	--	--

- 32 bit sums partial 16 bit products in MUL2

MULW	MUL0	MUL1	MUL2	ACC0	SHIFT0	SHIFT1	SHIFT3	WB
------	------	------	------	------	--------	--------	--------	----

MACW	MUL0	MUL1	MUL2	WB				
------	------	------	------	----	--	--	--	--

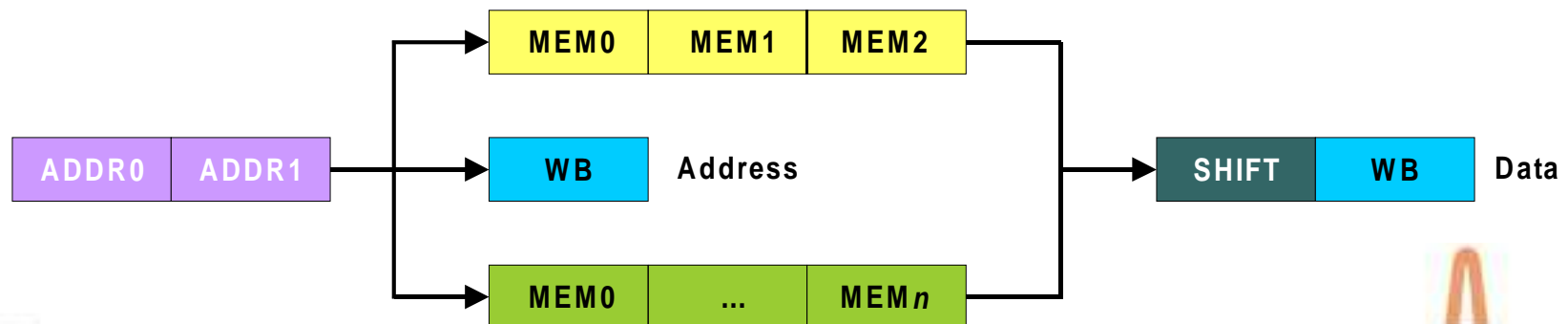
- Move

MMV				ACC0	SHIFT0	SHIFT1	SHIFT3	WB
-----	--	--	--	------	--------	--------	--------	----

# Load Store Pipeline

---

- Memory mapped into two regions
  - Local memory
  - Peripheral bus
- Address generated in first two cycles
  - Option of writing this value back
  - Instructions split to local memory or peripheral memory and index register writeback



# Load Store Pipeline local memory

---

- Local memory partitioned to enable pseudo dual port behaviour for concurrent accesses
  - Dual Load Store units produce two memory requests
  - These requests can both be misaligned
  - Code can be constructed so requests do not map to overlapping memory blocks
    - In the event of a clash one side is held back
  - No flow control beyond ADDR1 for local memory enables faster cycle times
  - Shifter aligns load data
    - Also sign extends signed loads

# Implementation Methodology

---

- 100% cell based design
  - In House standard cell library
  - Highly automatic library build
- Flexible implementation style
  - partial gate level design
  - partial preplacement
  - partial prerouting
- Uniformity where it counts
  - design captured in RTL
  - all netlists formally shown be equivalent