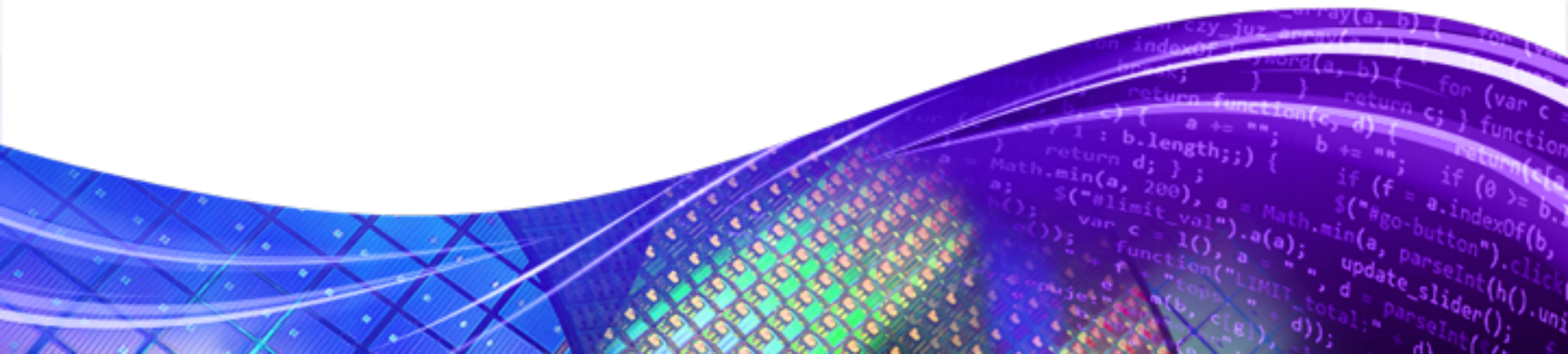
SYNOPSYS®
Silicon to Software™

Scheduling for ARC HS cores

GNU Cauldron 2015

Claudiu Zissulescu

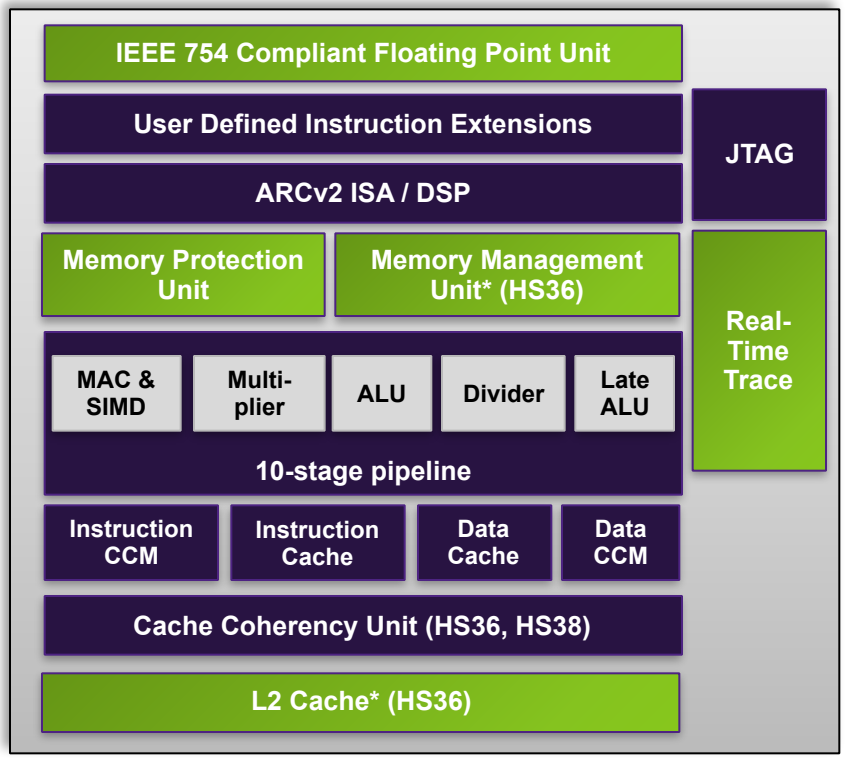


Overview

- ARC HS and its pipeline
- Schedule for HS
- Results
- Other features of GNU GCC compiler for ARC, and Conclusions

ARC HS Processor Family

High-Speed Multicore for High Performance Embedded



Licensable Option

*standard for HS38

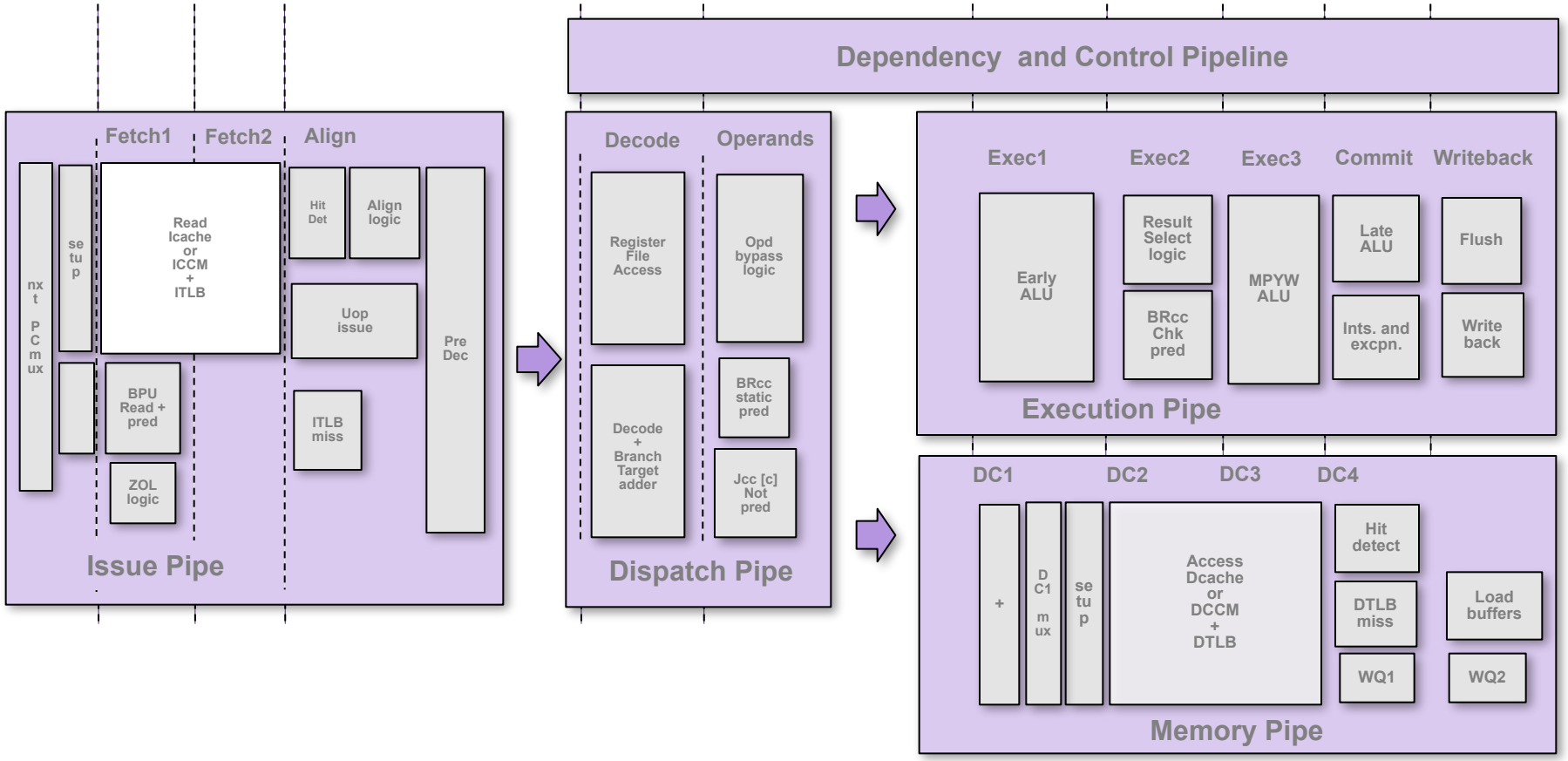
ARC HS34	16M I & D closely coupled memories (CCMs)
ARC HS36	*I & D CCMs, I & D caches (up to 64K ea)*
ARC HS38	I & D CCMs, I & D caches, L2 cache, MMU

- Highest performance ARC Processor family
  - Next-generation ARCv2 ISA
  - Over 3100 DMIPS @ 1.6 GHz, 53 mW power consumption, 0.12mm^2 area*
- Options include
  - IEEE 754 compliant FPU
  - MPU
  - Real Time Trace
  - MMU, L2 cache (standard on HS38)
- Single, dual and quad-core configurations

*28HPM, worst case silicon & conditions, ARC HS34

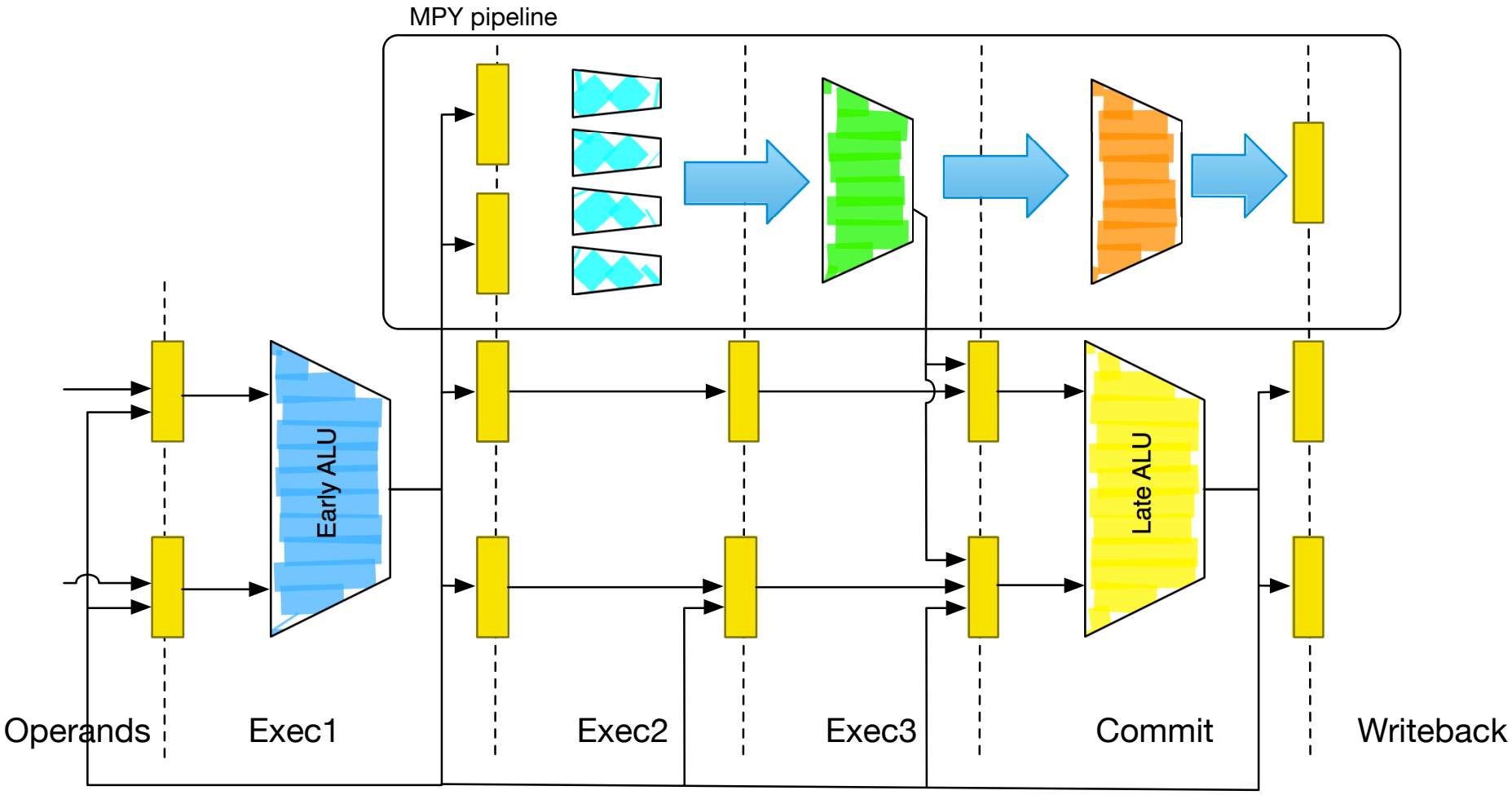
SYNOPSYS

Pipeline Overview



SYNOPSYS

ARC HS data path



SYNOPSYS

Early ALU exclusive instructions

- Simple ALU ops:
  - ADC, SBC, ABS, MIN, MAX, ADD1, ADD2, ADD3, SUB1, SUB2, SUB3
- Barrel Shifter ops:
  - ASR, ASL, LSR, ROR, ASR16, LSR16, LSL16, ASR8, LSR8, LSL8, ROR8, ROL8, XBFU
- SWAP ops:
  - SWAP, SWAPE
- Bit Scan ops:
  - NORM, NORMW, FFS, FLS
- Control ops:
  - LR, SR, SETI, CLRI, AEX, FLAG, KFLAG, LPcc, BI, BIH
- Multiply, division, MAC and vector operations starts one cycle later than Early ALU ops.

Schedule for HS: Idea

- Initialise the cost for all dependencies using a heuristic.

- Each time we schedule an new instruction:
  - Note on which ALU is the current instruction executed, by looking at the current instruction emulated clock and all its producers' clocks.
  - Recompute/Reset the cost for all remaining unscheduled TRUE-dependencies instructions accordingly.

Using GCC to schedule for ARC HS

- Each time when we start to schedule a function, we create for each instruction an instruction info structure holding the clock cycle and on which unit the instruction is scheduled (via TARGET_SCHED_INIT_GLOBAL). The clock cycle holds info when the instruction is scheduled.

- Using a heuristic, we initialise the cost of all the dependencies (via TARGET_SCHED_ADJUST_COST).

- Collect the current emulated scheduler clock via TARGET_SCHED_DFA_NEW_CYCLE.

- For each scheduled instruction (in TARGET_SCHED_VARIABLE_ISSUE), set the instruction clock (in insn_info) and compute what is the correct ALU on which this instruction is executed. Add this information to insn_info. Reset all the TRUE dependencies between this current instruction and its all forward dependencies. This re-triggers the call of the TARGET_SCHED_ADJUST_COST hook, that should (dynamically) update the invalidated DEP_COST.

Example

Before:

```
  7: r162:SI=[r158:SI]
  8: r161:SI=r162:SI+0xbe
  9: r163:SI=r159:SI<<0x2
 10: r164:SI=r163:SI+r159:SI
 11: r160:SI=r161:SI&r164:SI
 12: r157:SI=r160:SI
 16: r0:SI=r157:SI
```

After Sched2:

```
;;   0--> b  0: i   9r2=r1<<0x2      :(hs_issue+x1),x2
;;   1--> b  0: i   7r0=[r0]         :(hs_issue+hs_ld_st),hs_ld_st,nothing*2
;;   2--> b  0: i  10r1=r1+r2        :(hs_issue+x1),nothing*3
;;   3--> b  0: i   8r0=r0+0xbe      :(hs_issue+x1),nothing*3
;;   7--> b  0: i  16r0=r0&r1        :(hs_issue+x1),nothing*3

  9: r2:SI=r1:SI<<0x2
  7: r0:SI=[r0:SI]
 10: r1:SI=r1:SI+r2:SI
  8: r0:SI=r0:SI+0xbe
 16: r0:SI=r0:SI&r1:SI
```

Example: Pipe utilisation

	Operands	Exec1	Exec2	Exec3	Commit	Writeback
	asl_s r2,r1,2					
cycle 0	ld_s r0,[r0]	asl_s r2,r1,2				
cycle 1	add_s r1,r1,r2	ld_s r0,[r0]	asl_s r2,r1,2			
cycle 2	add r0,r0,190	add_s r1,r1,r2	ld_s r0,[r0]	asl_s r2,r1,2		
cycle 3	and_s r0,r0,r1	add r0,r0,190	add_s r1,r1,r2	ld_s r0,[r0]	asl_s r2,r1,2	
cycle 4		and_s r0,r0,r1	add r0,r0,190	add_s r1,r1,r2	ld_s r0,[r0]	asl_s r2,r1,2
cycle 5			and_s r0,r0,r1	add r0,r0,190	add_s r1,r1,r2	ld_s r0,[r0]
cycle 6				and_s r0,r0,r1	add r0,r0,190	add_s r1,r1,r2
cycle 7					and_s r0,r0,r1	add r0,r0,190
						and_s r0,r0,r1

Results

	GCC	In house compiler
Coremark	3.38	3.63
Dhrystone	1.89	1.92

Other ARC features supported by GCC

- IEEE 754 simple/double precision floating point instructions
- 64-bit operations
- Atomic operations
- SIMD operations, and autovectorization
- New code density operations

Conclusions

- The HS architecture has two ALUs: the Early ALU and the Late ALU. The Early ALU can execute all instructions, the Late ALU only a subset. The selection on which Early ALU a Late ALU instruction depends on the data readiness of its operands. An instruction executed on Early ALU has different bypasses than the very same executed on Late ALU.

- It will be nice to restart the entire scheduling process whenever we detect an instruction changing the current ALU assignment.

- It is still work in progress. We try to improve the model and make it more accurate. However, we haven't observed performance lost due to bad scheduling, yet.