UCLA COMPUTER SCIENCE DEPARTMENT

Center for Domain Specific Computing

Architecture Support for Custom Instructions with Memory Operations

Jason Cong and Karthik Gururaj

VLSI CADLAB

Department of Computer Science

University of California Los Angeles

Motivation

- Customized ISAs
  - Customized functional units
  - Generally read/write inputs/outputs from registers
  - Work great when customized for specific domains
    - ASIPs for video, audio, cryptography etc
  - Typically do not contain memory operations
- How does ISA customization work when:
  - Custom functional units are implemented on FPGA fabric
  - Coupled with a superscalar Out of Order (OoO) core pipeline
    - Cortex A9, A15, upcoming Atom Silvermont etc
  - Wider set of applications

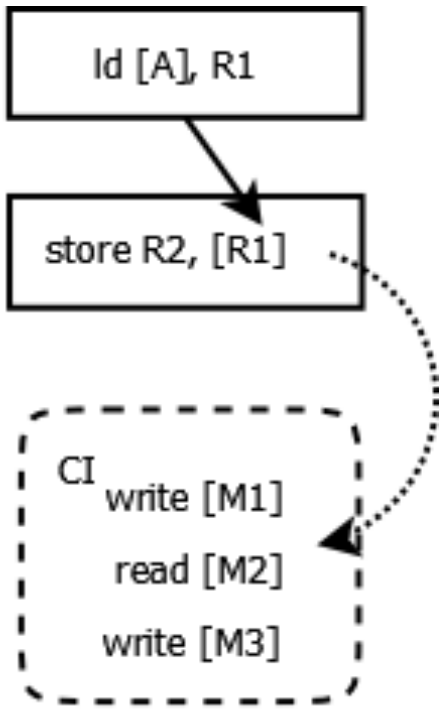
Motivation

- SPECint 2006
- No memory operations within custom instructions
- Performance improvement 1.4%
  - Energy reduction 5%

Benchmark	Performance improvement (%)	Energy reduction(%)
bzip2	0.9	0.76
mcf	0.2	1.2
gobmk	3.7	2.9
hmmer	5.3	8.4
sjeng	6.2	10.6
libquantum	-2.1	1.4
h264ref	-4.6	6.6
Average	1.45	5.18

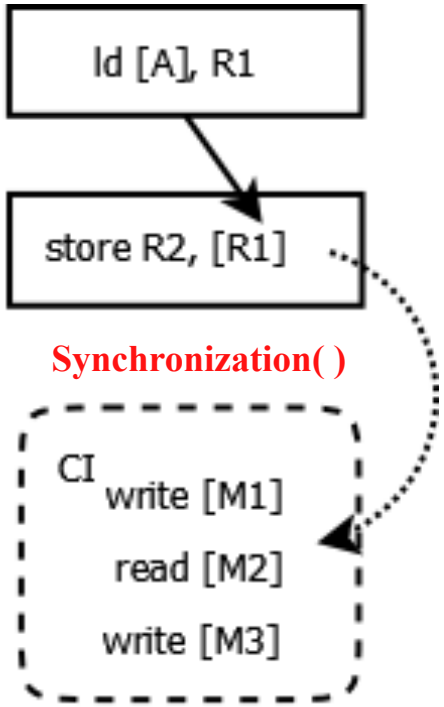
Motivation – example

- **Custom instruction – CI – with memory operations**
  - Challenge: ensure correct ordering of memory operations in an OoO pipeline
- **Possible dependence between *store* and *CI***
  - Depends on value of R1
  - Cannot be determined by compiler
- **Challenge**
  - How to ensure that program order is maintained?

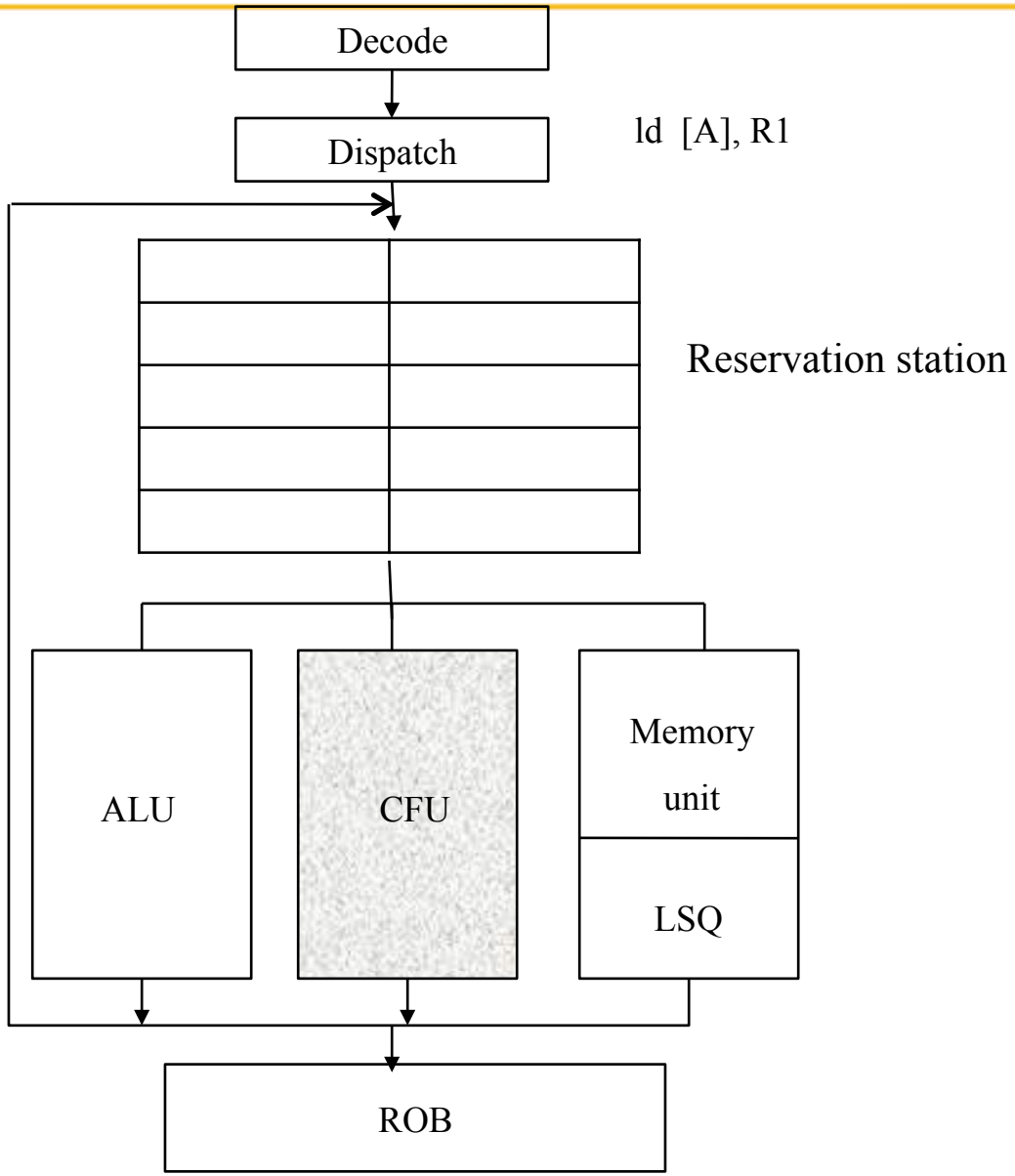


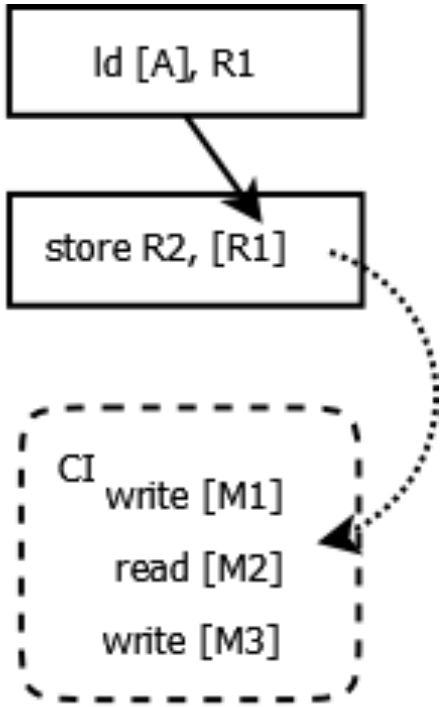
Motivation – example

- Previous work
  - One Chip [Carillo et al FPGA 2001]
  - Veal [Clark et al ISCA 2008]
  - DMA for ASIPs [Kluter et al MICRO 2009]
- Require the compiler to insert synchronization operations
  - Loss of performance – worse than software only
- Our contribution
  - Architecture and compiler support for custom instructions without using any synchronization

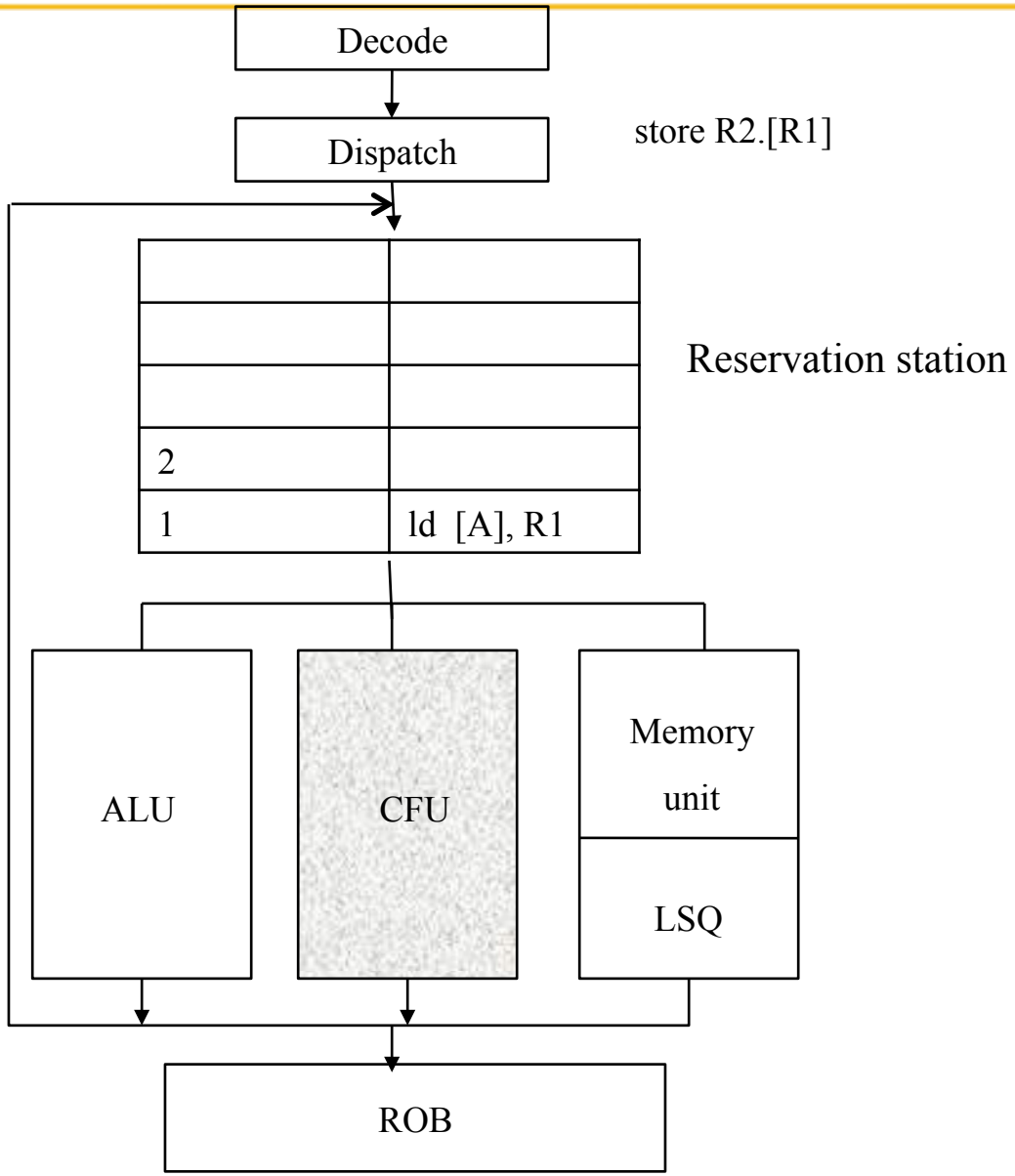


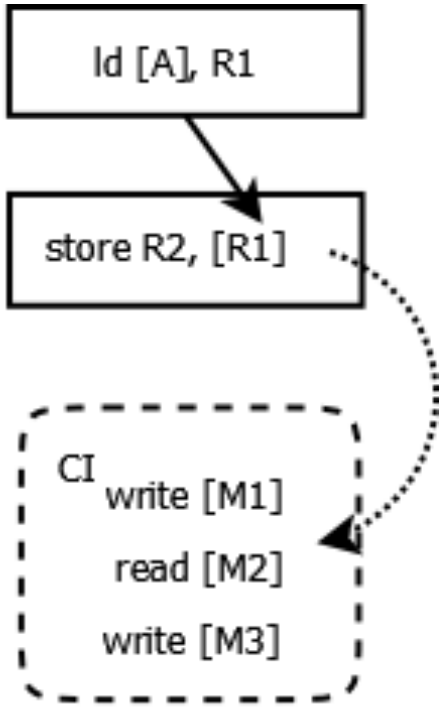
Typical OoO pipeline



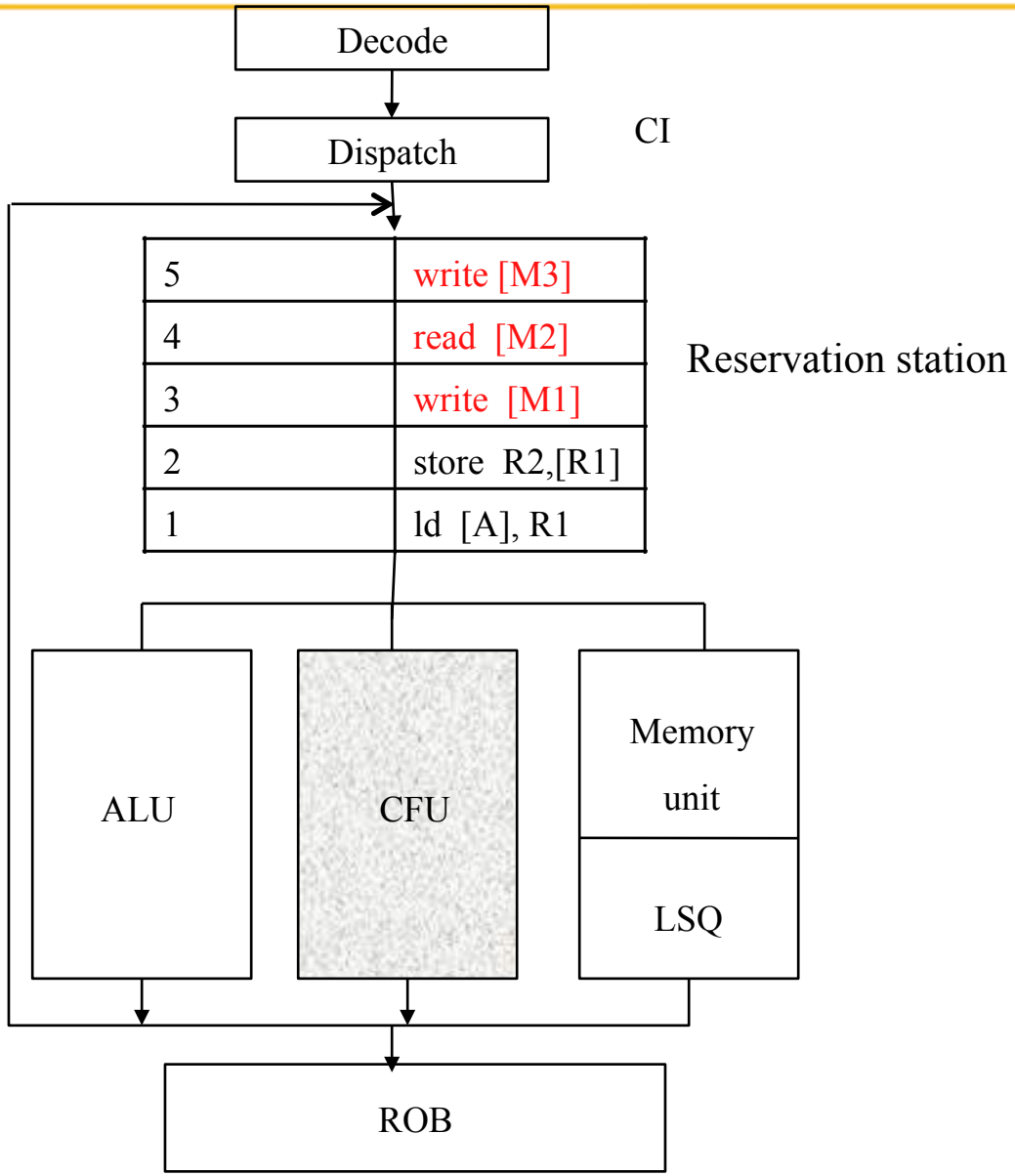


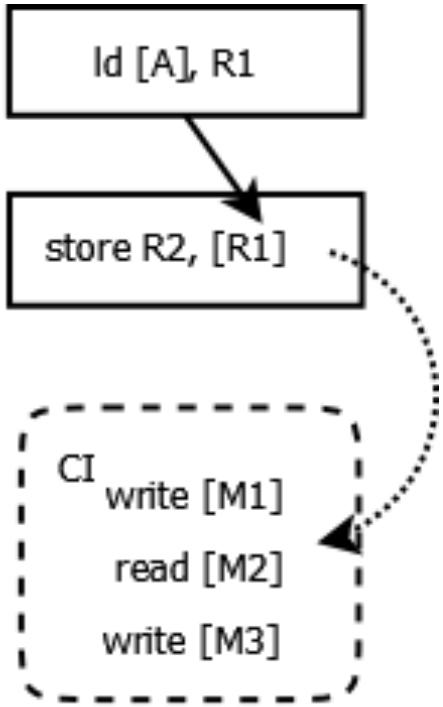
Pipeline snapshot



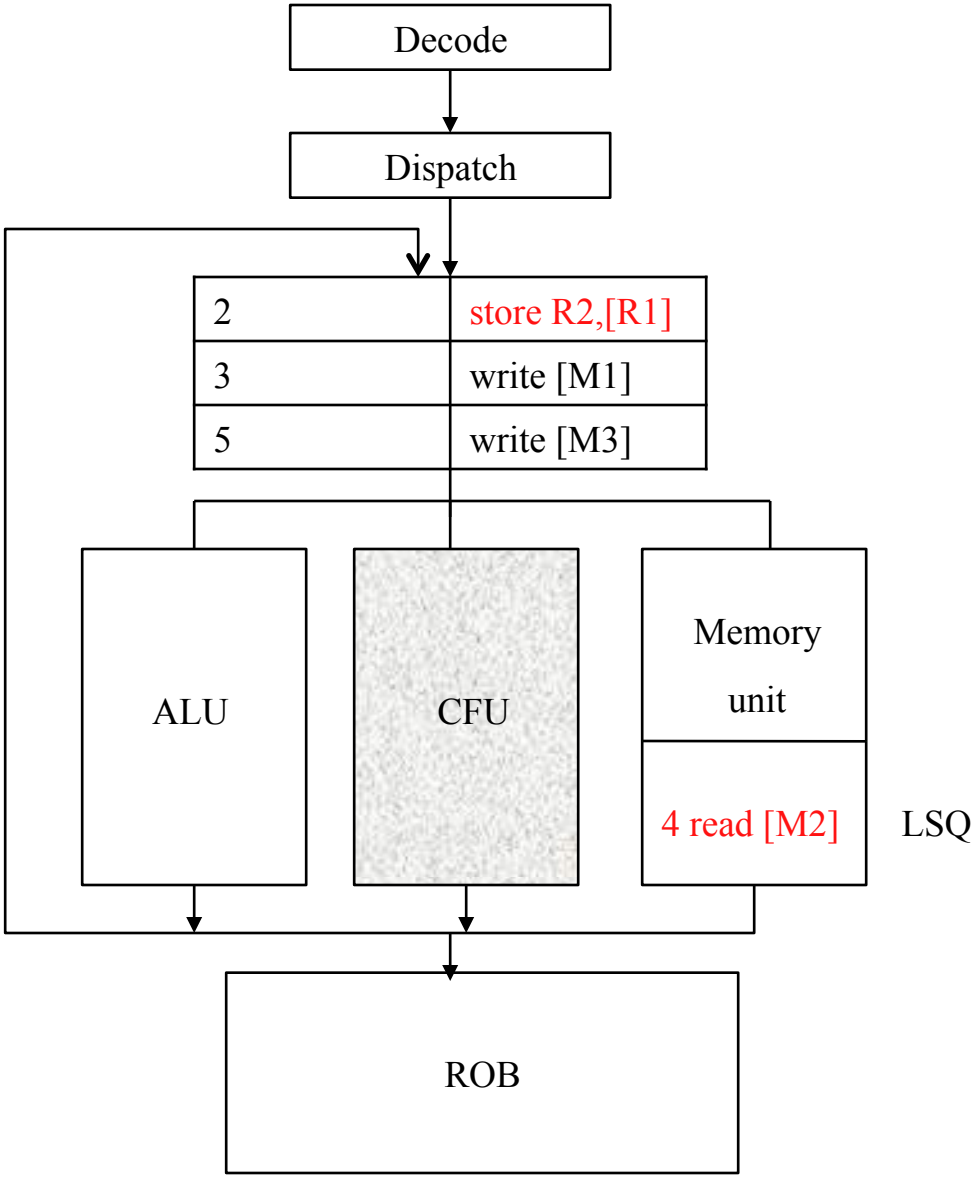


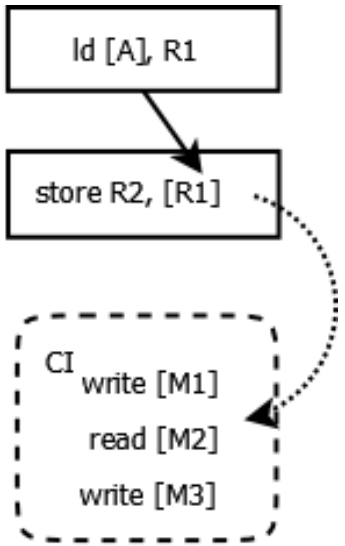
Pipeline snapshot



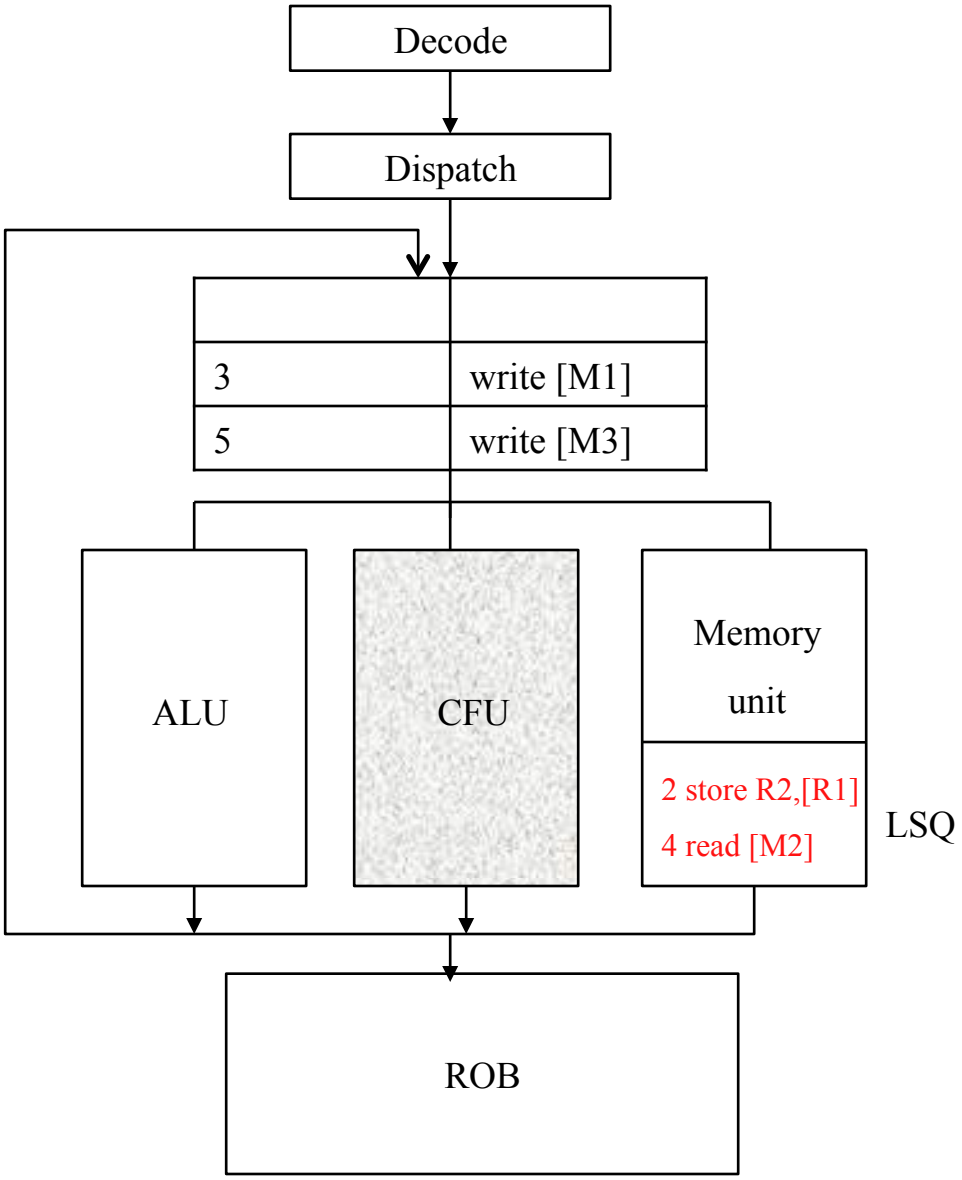


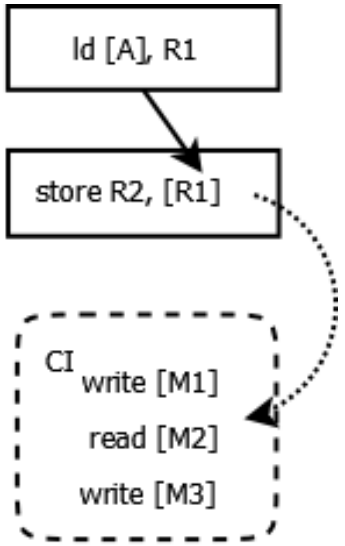
Pipeline snapshot





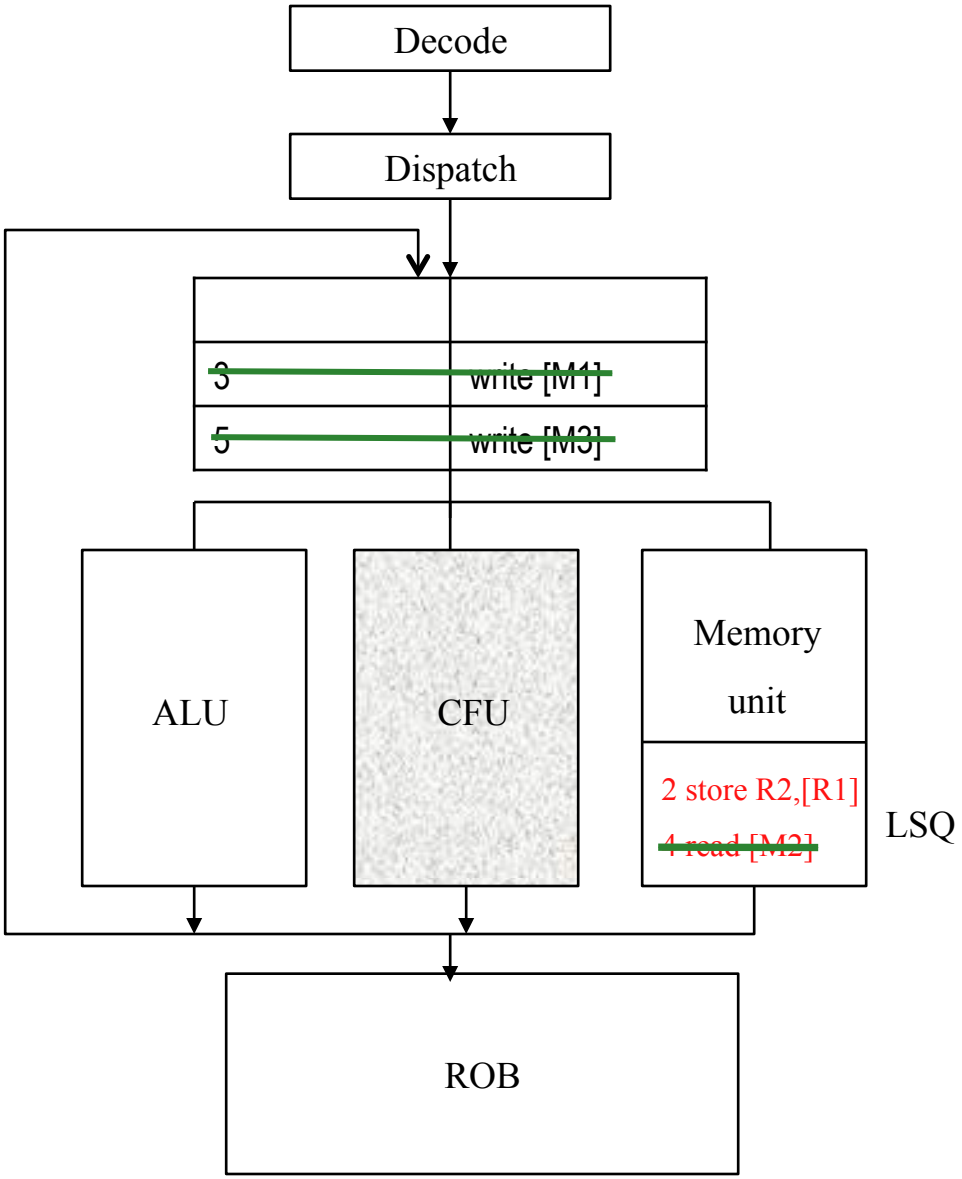
Pipeline snapshot – no dependence

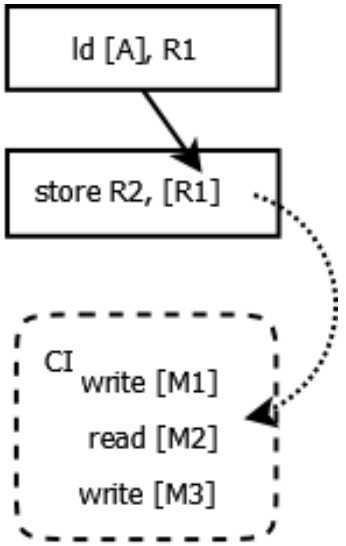




- If no dependence
  - Both operations proceed with execution
  - Retire eventually

Pipeline snapshot – with dependence





- If dependence exists
  - read [M2] squashed
  - Pipeline flush
- CI is re-executed

Results - performance

- **Average: 18.2 – 22% performance gain**

	2-issue/128 entries		2-issue/256 entries		4-issue/128 entries		4-issue/256 entries	
Performance	baseline	CFU	baseline	CFU	baseline	CFU	baseline	CFU
bzip2	1.000	0.945	0.999	0.944	0.958	0.901	0.956	0.902
libquantum	1.000	0.564	1.000	0.536	0.653	0.213	0.653	0.193
hmmer	1.000	0.714	1.000	0.701	0.775	0.483	0.775	0.472
mcf	1.000	0.964	1.000	0.963	0.973	0.937	0.973	0.936
gobmk	1.000	0.977	0.999	0.976	0.982	0.960	0.981	0.958
h264	1.000	0.693	0.998	0.700	0.765	0.465	0.763	0.447
sjeng	1.000	0.863	0.999	0.866	0.895	0.757	0.894	0.756
Average	1.000	0.817	0.999	0.812	0.857	0.674	0.857	0.666
Improvement(%)	--	18.298	--	18.722	--	21.401	--	22.202

Results - energy

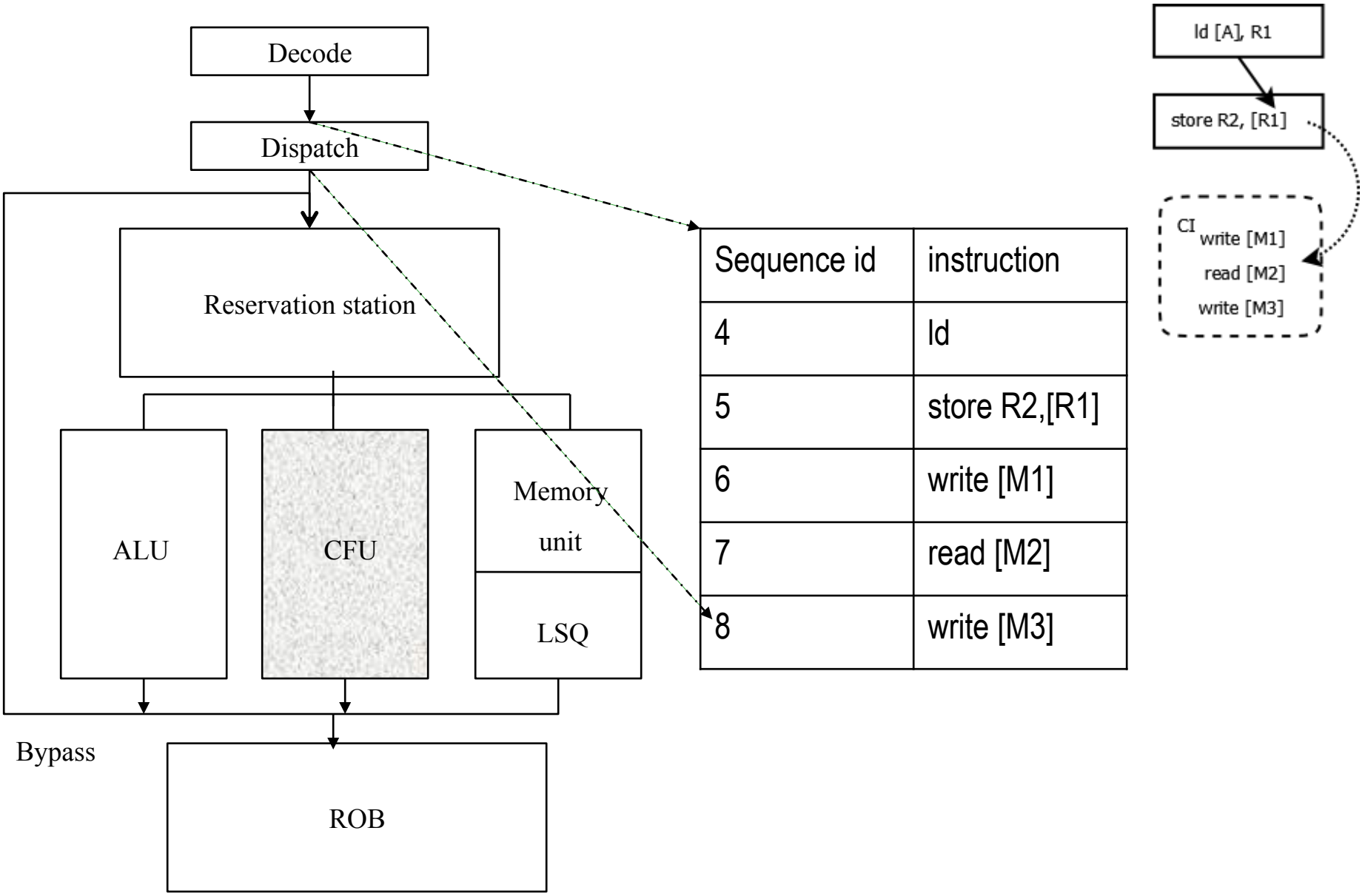
- **Average: 32 – 36% reduction**

	2-issue/128 entries		2-issue/256 entries		4-issue/128 entries		4-issue/256 entries	
	baseline	CFU	baseline	CFU	baseline	CFU	baseline	CFU
bzip2	1.000	0.691	1.046	0.716	1.367	0.949	1.452	0.929
libquantum	1.000	0.620	1.058	0.735	1.044	0.726	1.141	0.700
hmmer	1.000	0.615	1.094	0.743	1.079	0.685	1.291	0.888
mcf	1.000	0.698	1.060	0.723	1.035	0.717	1.123	0.711
gobmk	1.000	0.694	1.011	0.663	1.391	0.895	1.412	0.881
h264	1.000	0.678	1.051	0.657	1.137	0.718	1.224	0.768
sjeng	1.000	0.700	1.029	0.719	1.290	0.818	1.343	0.867
Average	1.000	0.671	1.050	0.708	1.192	0.787	1.284	0.821
Improvement(%)	--	32.894	--	32.582	--	33.986	--	36.075

Summary

- Architecture and compilation flow to ensure correct ordering of memory operations
  - In OoO pipelines
- Higher performance and energy savings

Our approach – sequencing memory operations



Sequence id	instruction
4	ld
5	store R2,[R1]
6	write [M1]
7	read [M2]
8	write [M3]