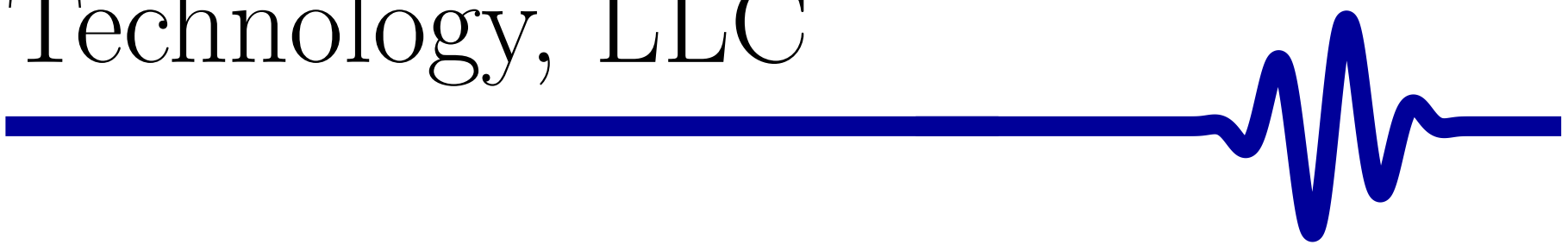
The ZipCPU

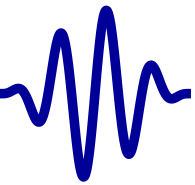
2017 Status Update

Gisselquist
Technology, LLC

Daniel E. Gisselquist, Ph.D.
September, 2017



ZipCPU Changes



- Now supports 8-bit bytes

  The wishbone select lines have been returned to the bus.

- Instruction set
  - New opcodes: `LW`, `LH`, `LB`, `SW`, `SH`, `SB`, ...

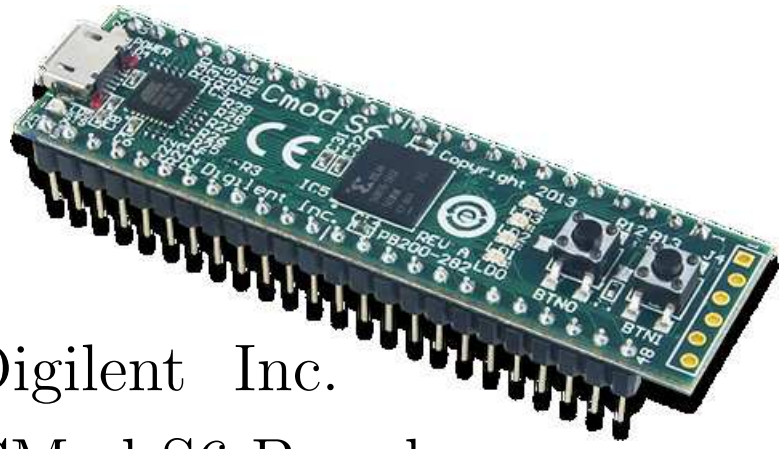
    These replace the old `LOD` and `STO` instructions.
  - Removed: rotate-left (`ROL`) and population count (`POPC`)
  - Compressed Instruction Set (CIS)

    Packs two 16-bit instructions into one 32-bit word.

- Double Prefetch
  - New optional low-logic pipelined fetch unit that fetches two instructions at a time

- Newlib now runs on the ZipCPU

Stressing Case



Digilent Inc.
CMod S6 Board

FPGA:	Spartan 6, LX4
Clock:	80 MHz
CPU:	Partial pipeline
LUTs:	2,400/2,400 (100%)
RAM:	4 kW
Flash:	4 MW

The processor now runs about 4x faster on the S6/LX4.

- Prefetch and instruction decode stages are now pipelined
- Flash is simpler, and runs at ~~40~~ 80 MHz SPI clock
- Prefetches two instructions at once (84 clocks → 29)
- 16-bit compressed instruction set (CIS) extension
- Can play 4x4x4 tic-tac-toe over UART using Newlib

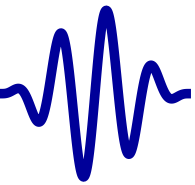
Survey of CPUs

Feature	NiOS	μBlaze	ECO–32	RISC-V	OpenRISC	LM32	ZipCPU
Open Architecture?	**No**		Yes				Yes
Number of Instructions	**86**	**129**	**61**	**50+**	**48+**	**62**	~~26+~~ 28+
OpCode Bits	6–17	6–11	6	10	6–32	6	5+
Interrupt/Exception Vectors	1	6	2	9+	14	32	None
Register Indirect plus displacement (bits)	16			12	16		14 ~~(18)~~
Immediate direct addressing (bits)	16, using R0=0						18 ~~(20)~~
Relative branching (bits)	16		26 (28)	21	26	21	18 ~~(20)~~
Conditional branching (bits)	16		16 (18)	13	26	16	18 ~~(20)~~
Register Size (bits)	32			32 (Opt. 64 Exts.)		32	32-bits
Special Purpose Registers	**6**	**25**	**6**	**66+**	**65+**	**10**	1 (x2)
General Purpose Registers	32 (but R0=0, others are unusable, . . . 24)						14 (x2)
8–bit data	Yes						~~No~~ Yes
16–bit data	Yes						~~No~~ Yes
32–bit data	Yes						Yes
64–bit data	No			Yes, by extension		No	Yes, not native
32–bit floats	Optional		No	Yes, by extension		No	Not yet
MMU	Yes, but optional						(Still in test)
Instruction Cache	Yes, configurable						Same
Data Cache	Yes, configurable						Not integrated

Survey of CPUs

Feature	NiOS	μBlaze	ECO–32	RISC-V	OpenRISC	LM32	ZipCPU
Open Architecture?	**No**		Yes				Yes
Number of Instructions	**86**	**129**	**61**	**50+**	**48+**	**62**	~~26+~~ 28+
OpCode Bits	6–17	6–11	6	10	6–32	6	5+
Interrupt/Exception Vectors	1	6	2	9+	14	32	None
Register Indirect plus displacement (bits)	16			12	16		14 ~~(18)~~
Immediate direct addressing (bits)	16, using R0=0						18 ~~(20)~~
Relative branching (bits)	16		26 (28)	21	26	21	18 ~~(20)~~
Conditional branching (bits)	16		16 (18)	13	26	16	18 ~~(20)~~
Register Size (bits)	32			32 (Opt. 64 Exts.)		32	32-bits
Special Purpose Registers	**6**	**25**	**6**	**66+**	**65+**	**10**	1 (x2)
General Purpose Registers	32 (but R0=0, others are unusable, . . . 24)						14 (x2)
8–	The big difference: 8-bit byte support		Yes				~~No~~ Yes
16			Yes				~~No~~ Yes
32			Yes				Yes
64			No	Yes, by extension		No	Yes, not native
32–bit floats	Optional		No	Yes, by extension		No	Not yet
MMU	Yes, but optional						(Still in test)
Instruction Cache	Yes, configurable						Same
Data Cache	Yes, configurable						Not integrated

The ZipCPU



Remains a simplified

- Simplified wishbone, instruction set, interrupt processing

open source,

- GNU General Public License (GPL), v3.0

low-area

- 1300-2600 LUTs

soft-core CPU?

Yes! The ZipCPU

Try it at: `https://github.com/ZipCPU/zipcpu` or
`http://opencores.org/project,zipcpu`

Gisselquist
Technology, LLC

In all labour there is profit . . .

Prov 14:23a

Backups

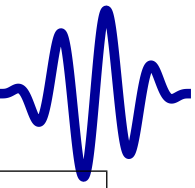
Performance

Xess
XuLA2-LX25

FPGA:	Spartan 6, LX25
Clock:	80 MHz
CPU:	All options on
LUTs:	7,735/15,032 (51%)
RAM:	8 kW
Flash:	256 kW
SDRAM:	8 MW

Other Peripherals: Serial port, PWM audio controller, Flash, SDRAM, SD Card controller, RTC clock, GPIO, and more.

Performance



Configuration	6-LUTs	ΔLUTs	CPU LUTs
S6SoC (w/CPU, no pipelining)	~~2345~~ 2400		
XuLA2 SoC, No CPU, Base System	~~2446~~ 2559		
CPU (no-pipeline, w/ debug)	~~3725~~ 3893	~~1286~~ 1344	[~~1286~~ 1344]
4-clock Multiply	~~3965~~ 4073	~~233~~ 180	[~~1519~~ 1514]
CPU (Double-fetch)	4051	(-22)	(1492)
CPU (Pipelined, 1kW I-Cache)	~~4383~~ 4446	~~418~~ 373	[~~1937~~ 1887]
Pipe Memory	~~4543~~ 4563	~~160~~ 117	[~~2097~~ 2004]
Early Branching	~~4541~~ 4543	~~-2~~ -20	[~~2095~~ 1984]
Divide *(Optimized out of Dhrystone Benchmark)*	~~4905~~ 4988	~~364~~ 445	[~~2459~~ 2429]
~~VLIW~~ CIS	~~5030~~ 5076	~~125~~ 88	[~~2584~~ 2517]
ZipSystem on XuLA2 board			
Basic (2x PIC, 3x timers, 2x watchdogs, jiffies)	~~5615~~ 5900	~~585~~ 824	[~~3169~~ 3341]
8x Performance Counters	~~6232~~ 6545	~~617~~ 645	[~~3786~~ 3986]
DMA	~~6882~~ 7216	~~650~~ 671	[~~4436~~ 4657]
XuLA2 Full up SoC (includes SD)	~~7372~~ 7735	~~490~~ 519	[~~4926~~ 5176]

32 OpCodes

ALU Instructions

SUB
AND
ADD
OR
XOR
LSR
LSL
ASR
BREV
LDILO
MPYUHI
MPYSHI
MPY
MOV
DIVU
DIVS

Divide unit

CMP
TEST
LW
SW
LH
SH
LB
SB
LDI

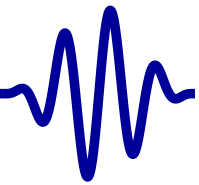
Memory operations

FPADD
FPSUB
FPMPY
FPDIV
FPI2F
FPF2I

/BREAK
/LOCK
/SIM
/NOOP

Reserved for floating point unit

From last year

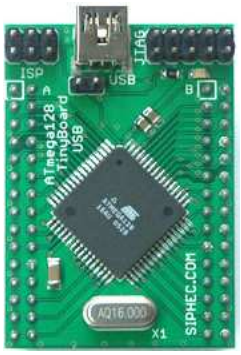


- Why do I need a ZipCPU?
- How has the ZipCPU been made resource efficient?
  - Simplified bus
  - Minimal instruction Set
  - A simpler approach to Interrupts
- Enhancements to the basic simplified ZipCPU
- What performance can be expected?

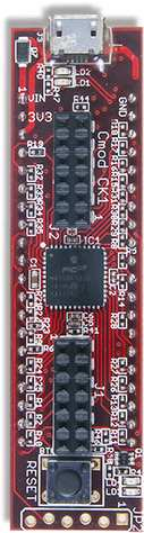
Vision: SwiC

If what you needed was a CPU, you would've bought one.

- All of the CPU's below are both *cheaper* and *faster*



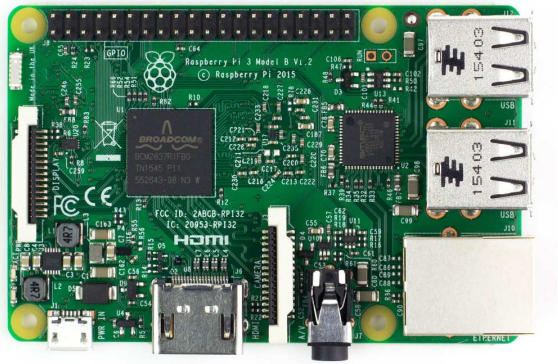
ATmega128



PIC32

MSP430

TeensyLC



RPi3/ARM

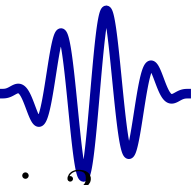
Vision: SwiC

But you bought an FPGA. Why?

- Because you had an application that needs lots of special purpose, high speed, processing to complete in time

Example: NetFPGA SUME

Vision: SwiC



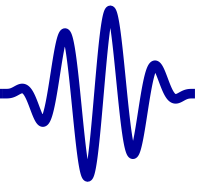
Does your application have a need for any sequential logic?

- Yes, but there's never enough room for it, and ...
- Both industry solutions, MicroBlaze and NiOS-II, would make your product vendor dependent
- What you need is a System within a Chip, or a SwiC!

This is therefore our goal and vision!

- A *small core* that can be added to a special purpose application, without drawing away too many resources
- An *Open Source core* than can be adapted to any vendor's hardware

Vision



Build a simplified, open source, low–area, soft-core CPU

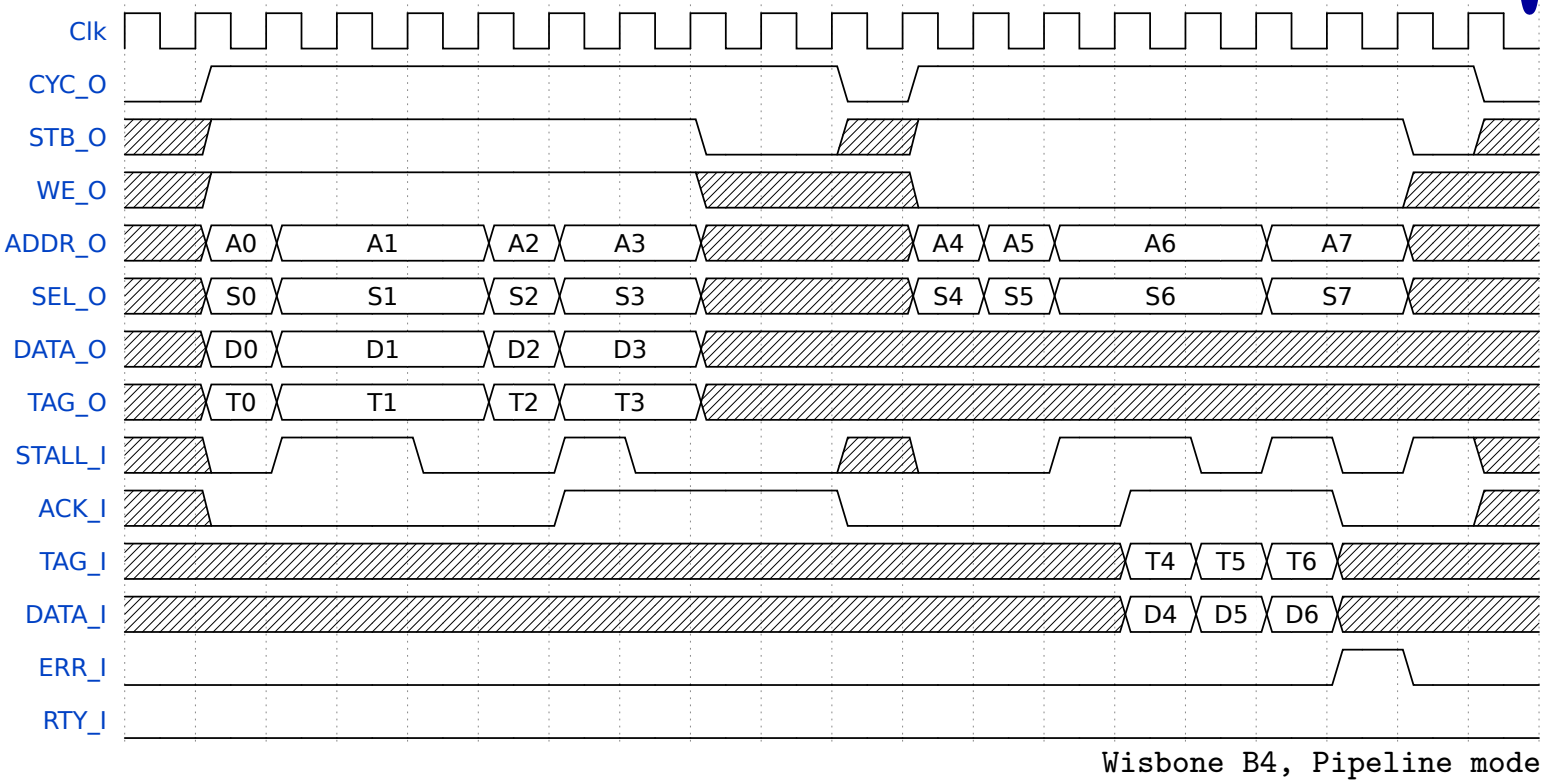
Goals

1. 32–bit
2. Pipelined
3. Wishbone
4. Threadable
   (Supervisor mode)

Choices

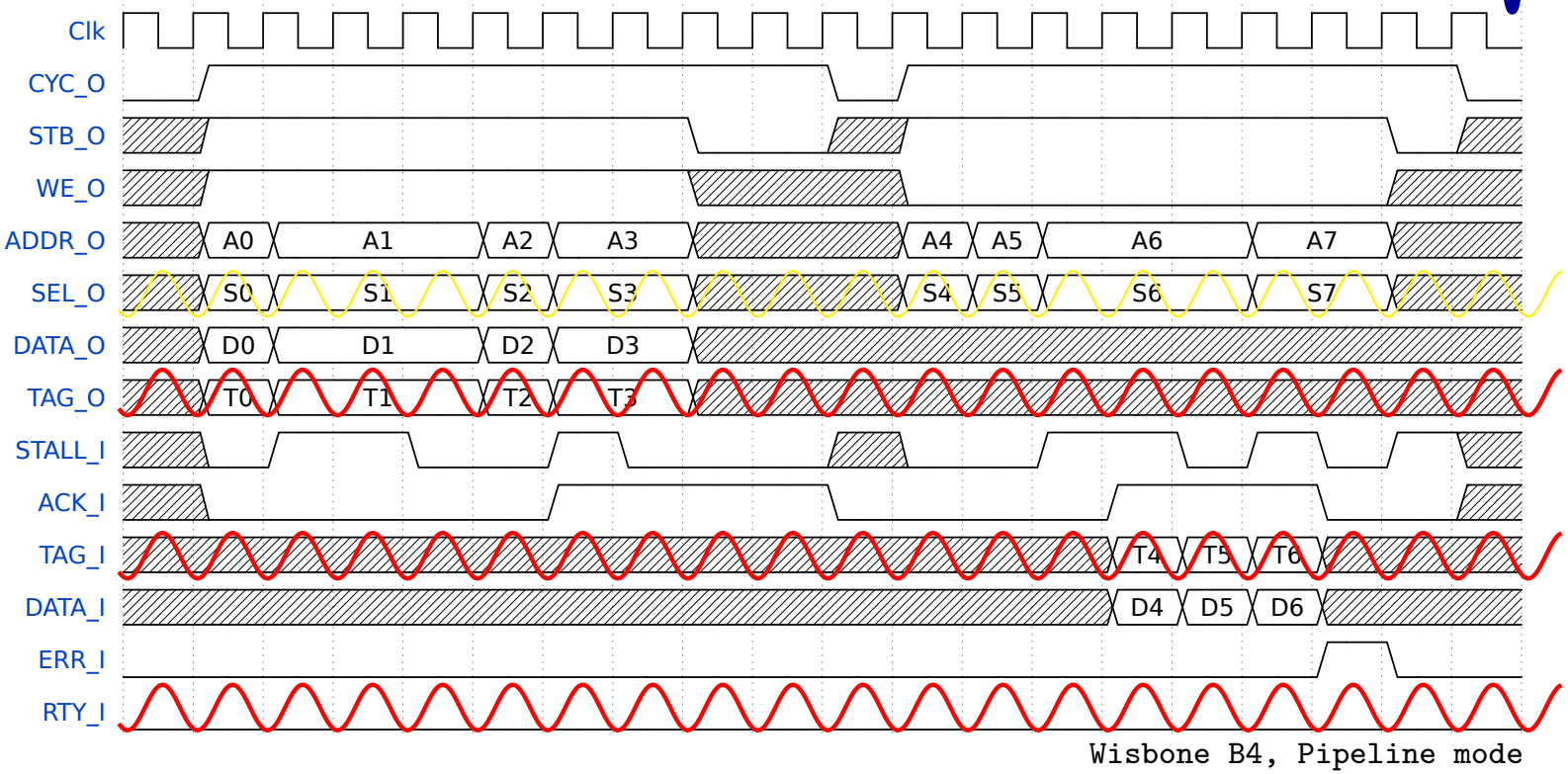
1. Simplified Wishbone
   - ~~Single word size: 32–bits~~
   - Only aligned accesses
   - Only one bus for I/D
2. Simplified instruction set
3. No interrupt vectors–interrupts just switch modes

Full Wishbone



Let's simplify this . . . can we remove anything we don't really need?

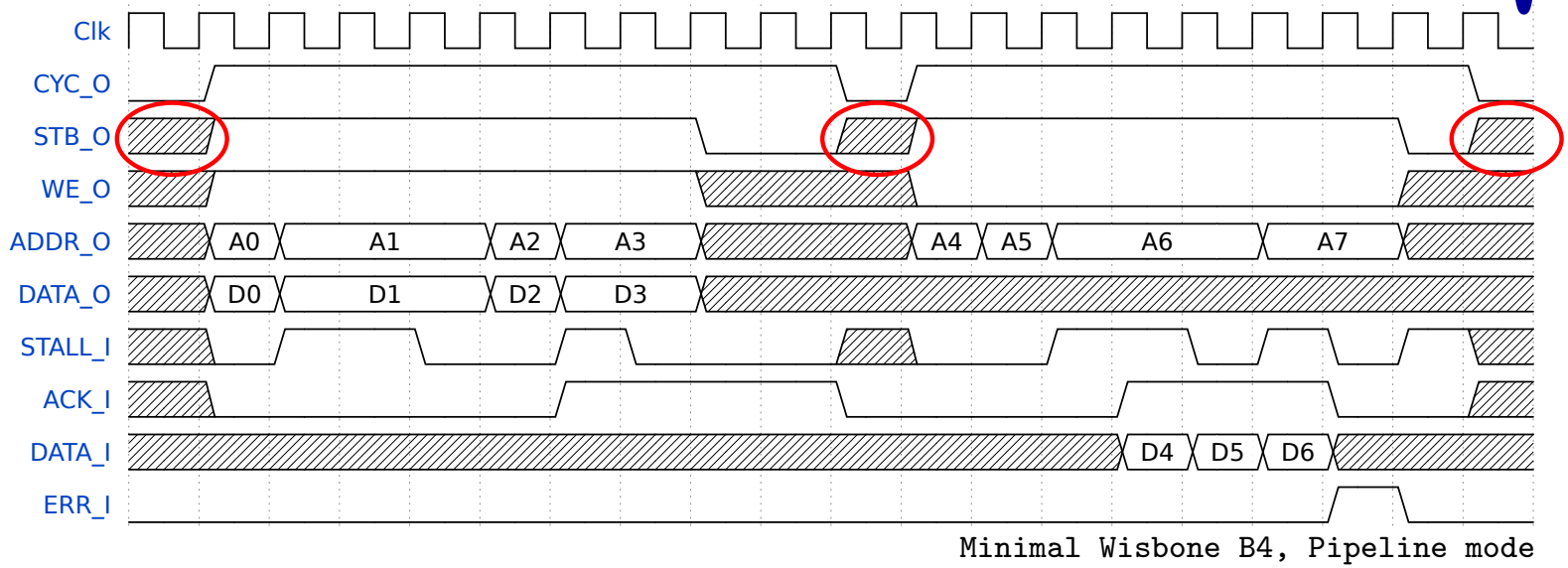
Simplified Wishbone



Let's remove the wires we don't need:

- Avoid anciliary information (TAGS, CTI_x)
- Merge the LOCK and CYC lines together
- ~~Force all transactions to be 32-bits, so remove SEL lines~~
- Ignore retries (RTY), we weren't using them anyway

Simplified Wishbone



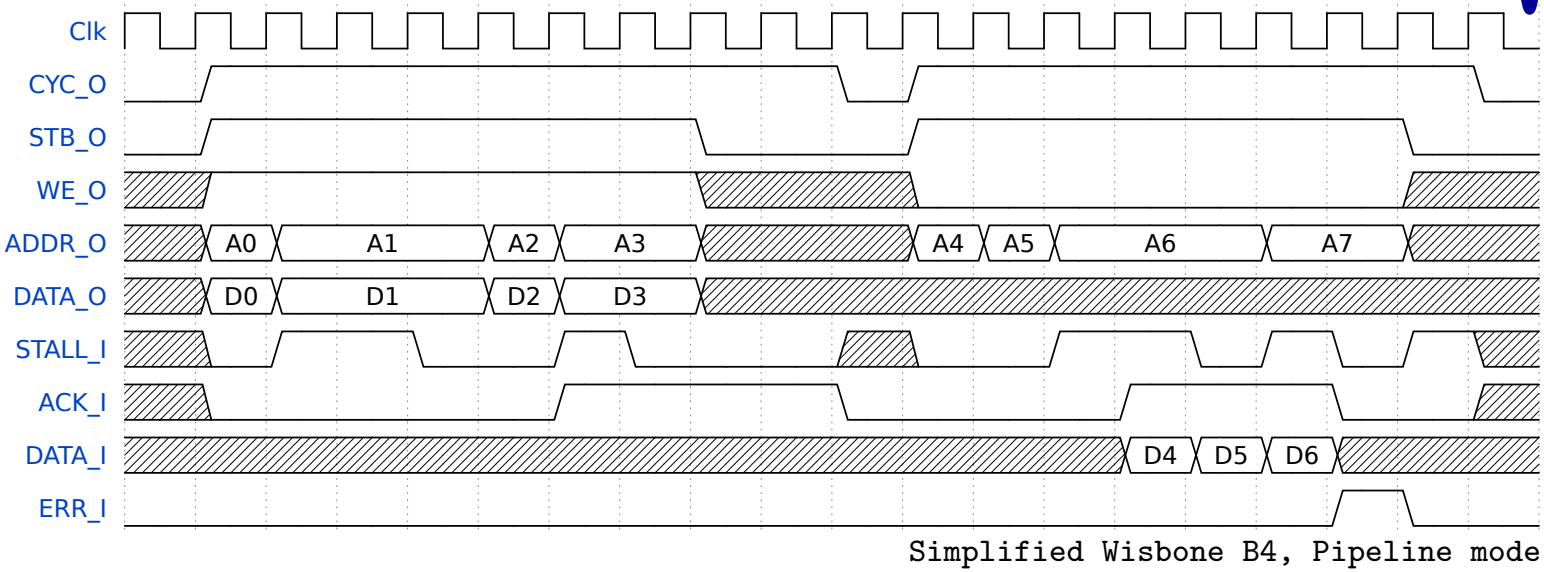
Let's simplify our remaining logic:

> Insist that `STB` be zero, rather than don't care, if `CYC` is zero

This simplifies a slave's decode logic: `if (CYC)&&(STB)` becomes `if (STB)` in any bus slave/peripheral.

I would recommend this change to the Wishbone standards body.

Simplified Wishbone



Simplified Wisbone B4, Pipeline mode

Transaction is complete when `(CYC)` returns to zero

- Bus is idle after the last `ACK`

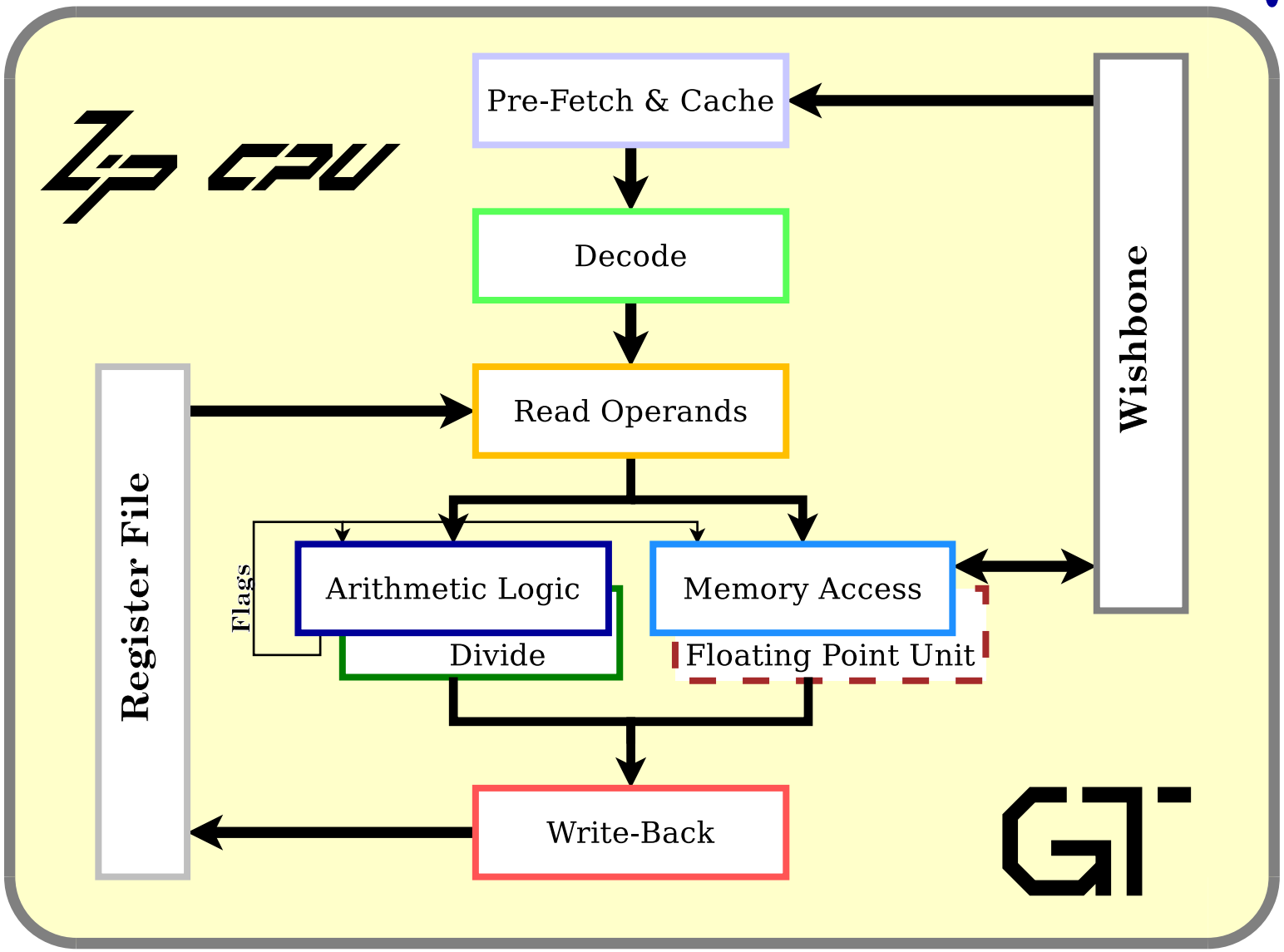
`(STB)&&(!STALL)` implies a transaction request took place

- Devices that don't stall only need to check `STB`
- Master must set check both `STB` and `STALL`

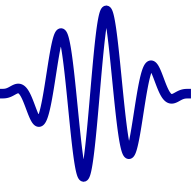
Every request expects an `ACK`

All transactions use pipeline mode

CPU Structure



Register Set

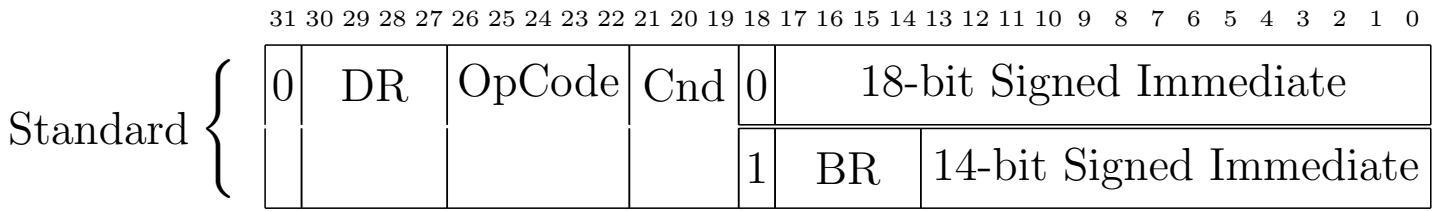


Two register sets, only one set is active at any time

Supervisor Register Set			User Register Set	
sR0(LR)	sR8		uR0(LR)	uR8
sR1	sR9		uR1	uR9
sR2	sR10		uR2	uR10
sR3	sR11		uR3	uR11
sR4	sR12(FP)		uR4	uR12(FP)
sR5	sSP		uR5	uSP
sR6	sCC		uR6	uCC
sR7	sPC		uR7	uPC
Interrupts Disabled			Interrupts Enabled	

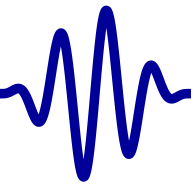
- Only the PC/CC registers have any special H/W purpose
- A special `MOV` instruction provides access to user regs

Simplified Insns



- 5–bit OpCode allows for 32 instructions
- 3–bit Condition allows every instruction to be conditional
- 4–bit Register code allows for up to 16 registers
- All instructions take either one or two registers
  - `OP.C #X+Rb,Ra`
  - `OP.C #X,Ra`
- This works until
  - the supervisor needs access to the user registers, or
  - you want to load a large number into a register

Notable



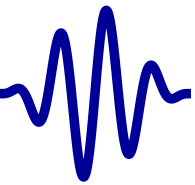
Unusual Instructions

1. `BREV` (bit reverse)
2. `TEST` (`AND` sets cond)
3. `CMP.`x (sets cond if X)
4. ~~`ROL` (rotate left)~~
5. ~~`POPC` (pop count)~~
6. `LDILO` (Load imm, lo)
7. `LOCK` (for atomic access)
8. `BRA` (`ADD #x,PC`)

"Missing" Instructions

1. ~~`LB`, `SB`, `LH`, `SH`~~
2. `PUSH`, `POP`
3. `CALL`, `JSR`, `JAL`, `JALR`
4. `RET`urn
5. `ADDC`, `SUBC`, `SUBR`
6. Compare and branch
7. Shift w/ carry
8. Compare and set
9. Set if zero

8 Conditions



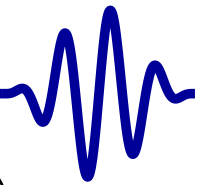
ZipCPU supports eight conditions:

Code	Meaning	CC Bits	Usage
3'b000	(Always)		
3'b001	.Z	Z	$A = B$
3'b010	.LT	N	$A < B$ (signed)
3'b011	.C	C	$A < B$ (unsigned)
3'b100	.V	V	On overflow
3'b101	.NZ	(!Z)	$A \neq B$
3'b110	.GE	(!N)	$A \geq B$ (signed)
3'b111	.NC	(!C)	$A \geq B$ (unsigned)

Updated in 2017

Any instruction can be executed conditionally

Function Calls



There are no subroutine OpCodes (`JSR`, `JAL`, `JALR`, etc.)

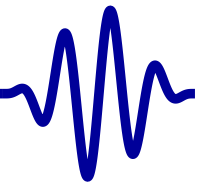
- Such instructions require two writes to the register set: one to store the PC, one to set the PC
- Solution: Function calls just take an extra instruction

```
    MOV return_lbl(PC),R0  ; This is the link instruction
    BRA subroutine         ; Implemented as an ADD #x,PC
return_lbl:
```

- Returns are simply indirect jumps

```
    JMP R0 ; Indirect branches cost 6-cycles
           ; Implemented as a MOV R0,PC
```

Interrupts



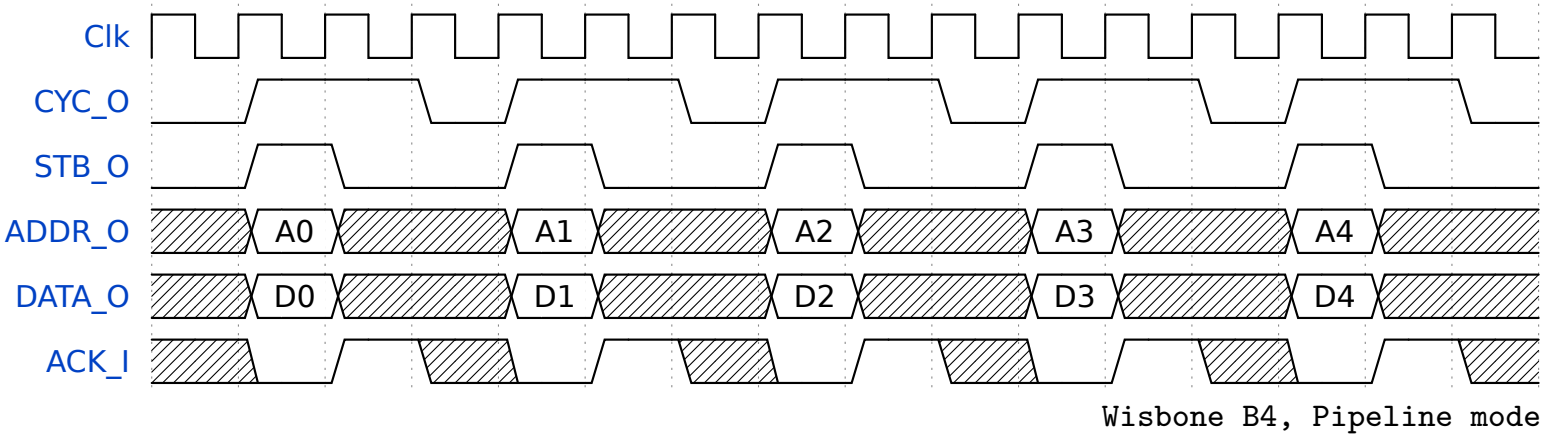
ZipCPU's approach to interrupts is . . . different:

- Only one interrupt line to the CPU
- No interrupt vectors, tables or "handler" functions
- ZipCPU just switches from user to supervisor mode

```
void entry(void) { // Supervisor entry function, on CPU reset
    // Setup up user tasks
    while(1) {
        zip_rtu(); // Return to userspace instruction
                   // Equivalent to OR #0x20,CC, sets GIE bit
        // Handle interrupts, traps and exceptions
        // Run scheduler, swap contexts?
    }
}
```

Pipelined Memory

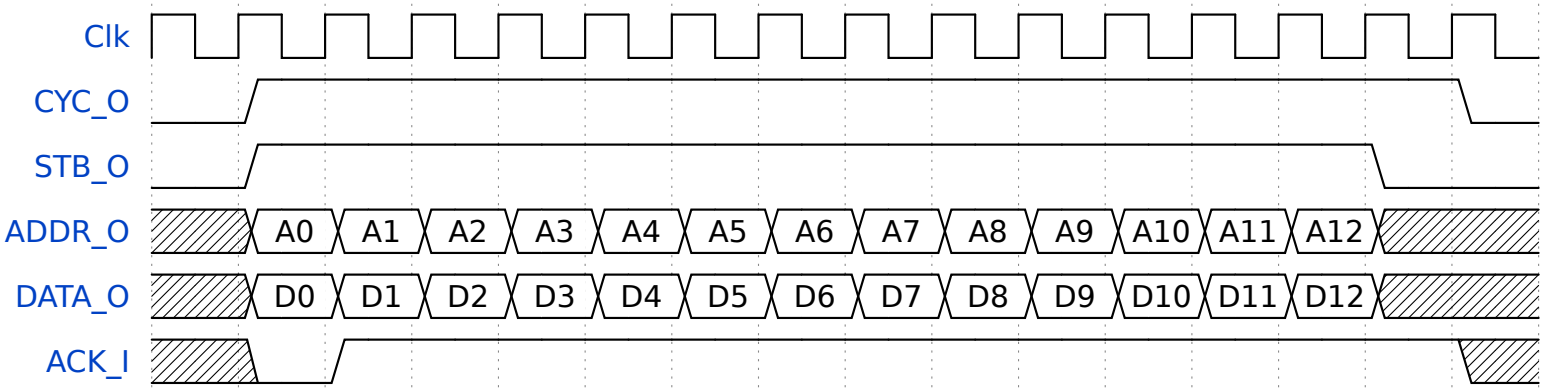
Without pipelining:



Best case transfer time: $3N$ clocks

Pipelined Memory

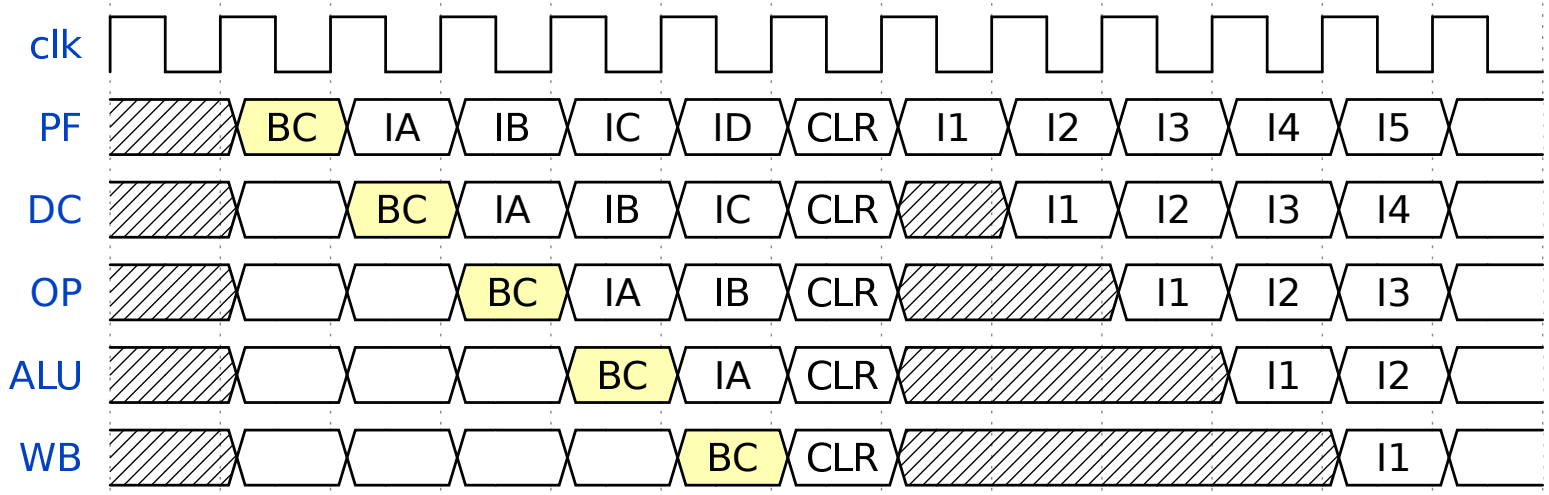
With pipelining:



Best case transfer time: $N + 2$ clocks

ZipCPU supports pipelined `LOD` and `STO` instructions

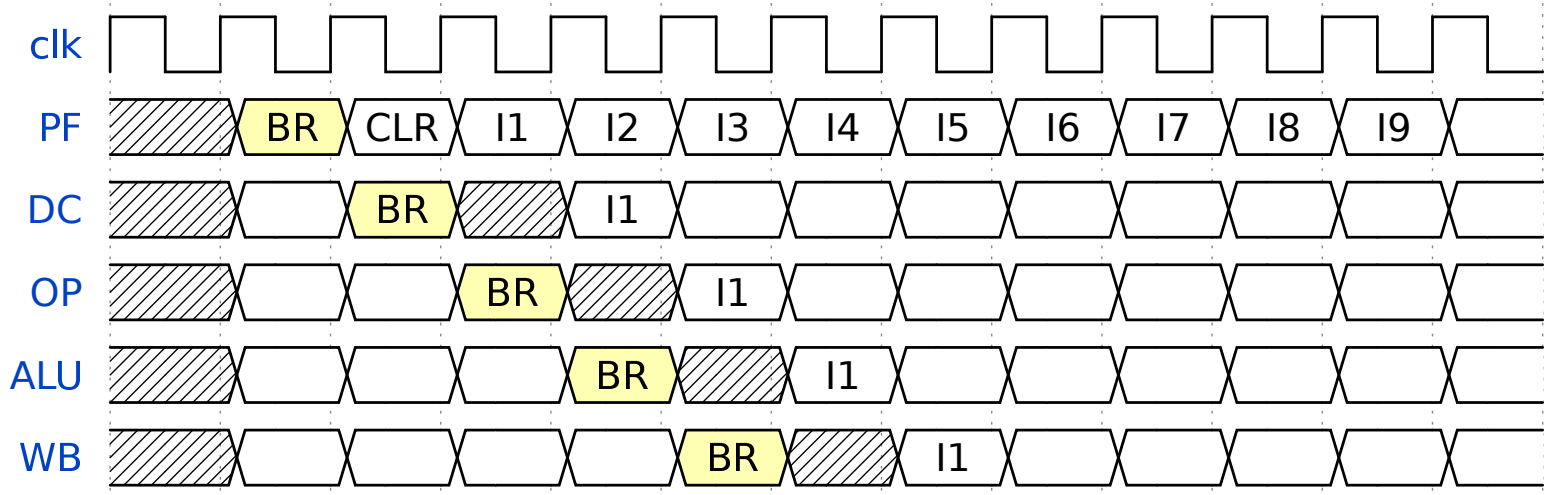
Early Branching



Instruction cost: 6 clocks

Without special logic, branches cost a full pipeline stall

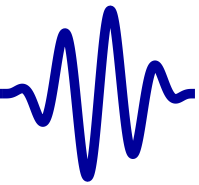
Early Branching



Instruction cost: 2 clocks

Early branching allows early detection of unconditional branch instructions, before the whole pipeline needs to be cleared.

Early Branching



- Three early branching instructions supported

  `LDI #x,PC` 2–cycles

  `ADD #x,PC` (`BRA`) 2–cycles

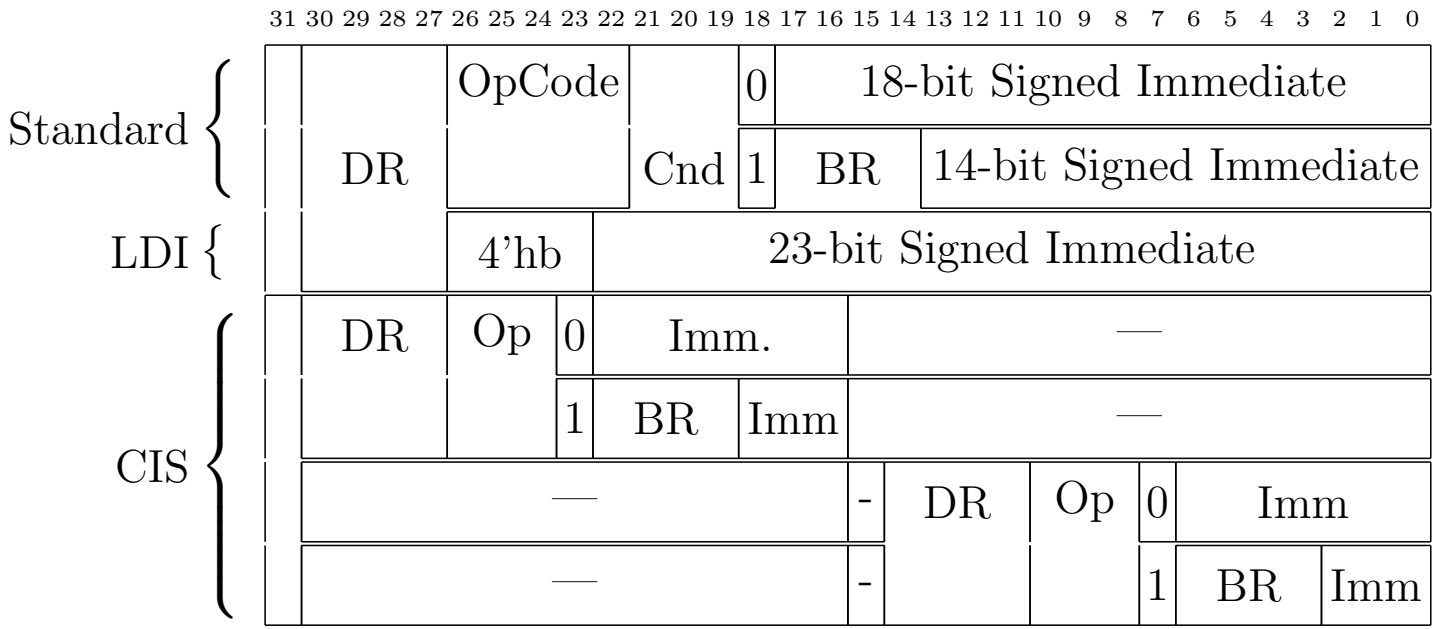
  `LOD (PC),PC` (`LJMP`) 3–cycles

  `LJMP` *was an afterthought to support linking*

- `LDI #x,PC` is hardly ever used
  - The address must fit inside 23–bits (signed)
  - The decision of which instruction (`LDI` vs `LJMP`) is made before the absolute address is known

    Perhaps I should recover this unused logic . . .

CIS Instructions



CIS instructions pack two instructions into one word, at the cost of a smaller immediate range.