FPGA soft-core

Jan Mennekens
UWINLOC
www.uwinloc.com

Thomas Moore — 16 mar 2018

learning goals

- soft cores : advantages and limitations
- design techniques
- debugging techniques
- best practices
- practical python/MyHDL use

soft-core

- micro blaze (xilinx) / Nios (Altera) / RISC-V (everybody)
- different from embedded ARM/PowerPC cores
- ASIP : Application Specific Instruction Processor
- FPGA != ASIC

when to use a soft-core

use an ASIP if you have a

- very specific application : high speed coprocessing, …
- customisable state-machine type solution

but DON'T use it if you need

- sophisticated programming in a higher language
- low-power
- very low cost

choices to make

- RISC / CISC / stack / other
- execution speed vs complexity
- memory size
- specialised instructions?

our core design goals

- small
- very simple
- extensible

design decisions

- stack processor
- shallow stack
- 5-bit instructions
- 12-bit registers
- up to 4k instructions program memory
- up to 4k words data memory
- memory-mapped I/O

ISA

5	3	2	1	0
INSTR	**OPCODE**			
1	LOAD OPCODE as CONSTANT in A			
0	EXECUTE OPCODE			

opcodes

OPCODE	MNEMONIC	FUNCTION	DETAILS
0	NOT	one's complement	A = ~A
1	NEG	two's complement	A = -A
2	SHL	shift left	A << 1
3	SHR	shift right	A >> 1
4	DROP	drop value	A = B, B = C
5	DUP	duplicate value	C = B, B = A
6	OVER	copy value over	A = C, B = A, C = B
7	XOR	exclusive OR	A = A ^ B
8	AND	AND	A = A & B
9	ADD	add	A = A + B
10	RMEM	read memory	A = MEM[A]
11	WMEM	write memory	MEM[A} = B, A = C, B = C
12	CALL	call subroutine	I = A, A = I + 1
13	JUMP	jump to location	I = I + A
14	SKIP	A = C; B = C; if B == 0, skip A instructions	
15	HALT	:w	meta-instruction

extensible constant

#0x4	A = 0x004
#0x8	A = 0x048
ADD	A = ????
#0xD	A = 0x0D

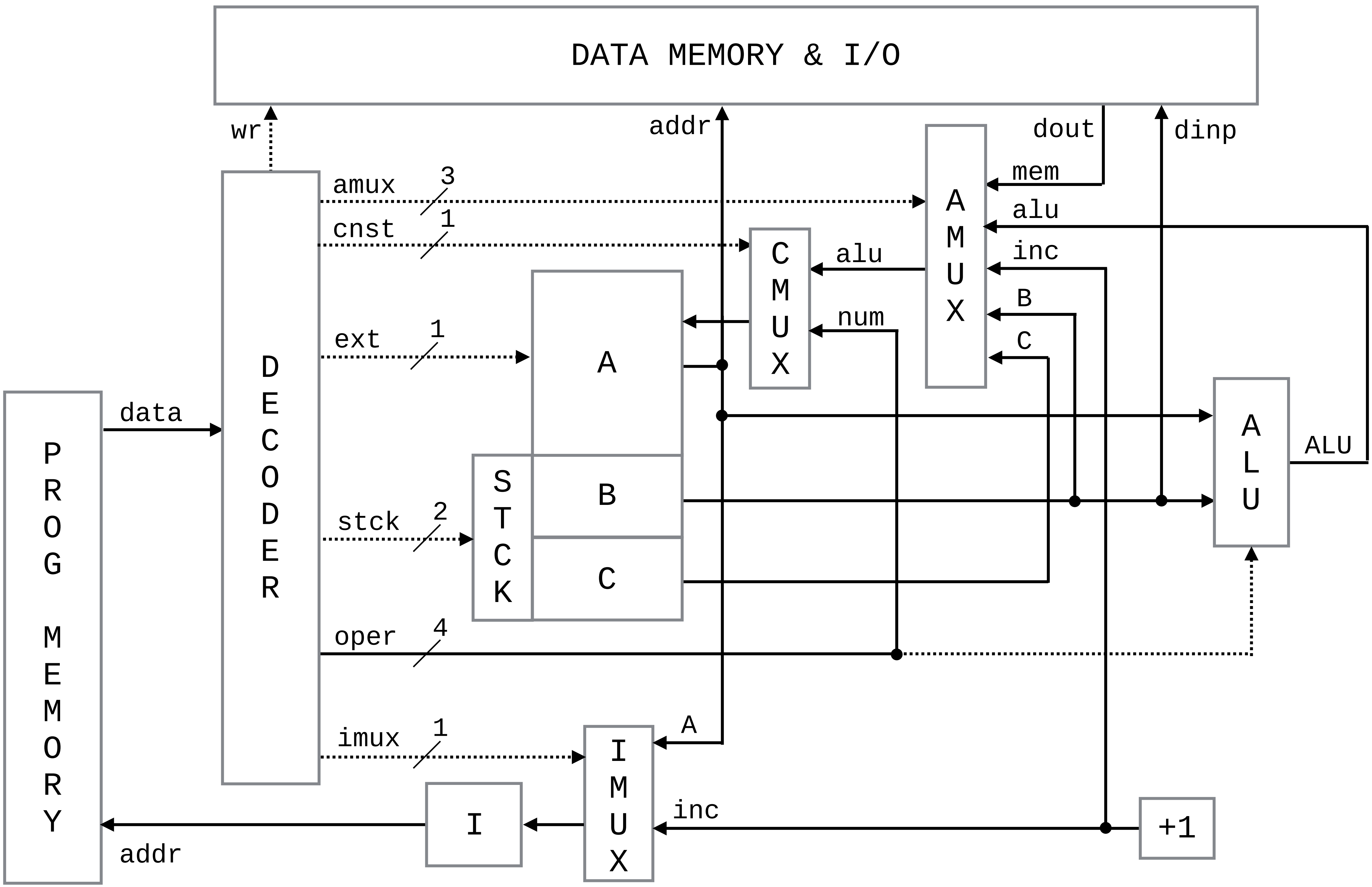
- higher code density
- binary code compatibility across register sizes
- no register/code size limit

SKIP instruction

if (B), skip next A instructions

- better pipelining
- more flexible coding

stack processor



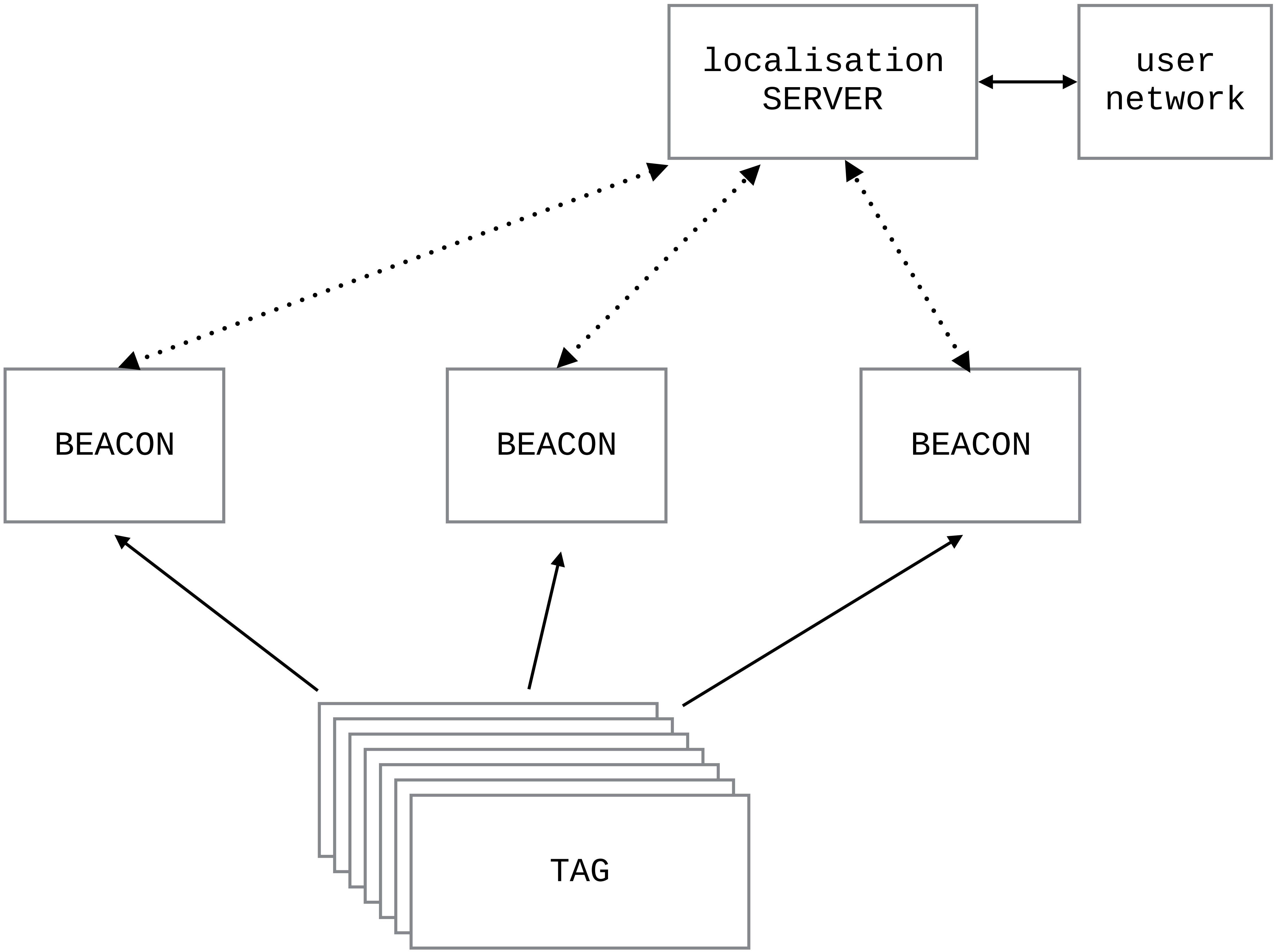
controls

FUNCTION	ALU (4)	STCK (2)	AMUX (2)	IMUX (1)	WR (1)
NOP	A (0)	nop (0)	alu (0)	inc (0)	no (0)
NOT	!A (1)	nop (0)	alu (0)	inc (0)	no (0)
NEG	-A (2)	nop (0)	alu (0)	inc (0)	no (0)
SHL	A << 1 (3)	nop (0)	alu (0)	inc (0)	no (0)
SHR	A >> 1 (4)	nop (0)	alu (0)	inc (0)	no (0)
READ	A (0)	nop (2)	mem (1)	inc (0)	no (0)
DUP	A (0)	push (1)	alu (0)	inc (0)	no (0)
—-	unused	nop (0)	alu (0)	inc (0)	no (0)
—-	unused	pop (2)	alu (0)	inc (0)	no (0)
XOR	A ^ B (5)	pop (2)	alu (0)	inc (0)	no (0)
AND	A & B (6)	pop (2)	alu (0)	inc (0)	no (0)
ADD	A + B (7)	pop (2)	alu (0)	inc (0)	no (0)
CALL	A (0)	pop (2)	inc (2)	alu (1)	no (0)
WRITE	A (0)	pop (2)	C (3)	inc (0)	yes (1)
IFEQ	A (0)	pop (2)	alu (0)	inc (0)	no (0)
IFLT	A (0)	pop (2)	alu (0)	inc (0)	no (0)

UWINLOC

commercial break

UWINLOC system



UWINLOC data flow