A possible evolution of the ‘MCPU’ 32 macrocell CPLD CPU architecture

ANDREW BURGE

V2.0, 25/APRIL/2021

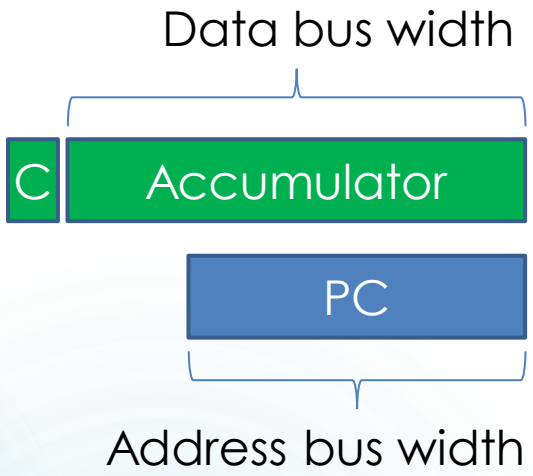
MCPU - The starting point

MCPU was a 2001 university project to implement a simple CPU in a 32 macrocell CPLD

Originally published via the university but now a small GitHub project

- MCPU has just **4** instructions!
  - All other instructions must be synthesised from sequences of these 4!!
- MCPU uses a memory / accumulator architecture
  - The prior MPROZ design uses a memory / memory architecture
    - MPROZ requires one less instruction than MCPU but the code density is inferior
- MCPU was implemented in VHDL, with a Verilog example provided

MCPU Instruction Set Architecture



There are two registers in MPCU:

The Accumulator
The Program Counter

All instructions use exactly one machine word and have a simple, fixed, coding.

The example shows an 8 bit data bus with 2 instruction bits decoding 1 of 4 instructions.
Thus the PC and address bus are both 6 bits wide.


```
NOR    0 0 A A A A A A    Accu = Accu NOR mem[AAAAAA]
ADD    0 1 A A A A A A    Accu = Accu  +  mem[AAAAAA]; Update carry
STA    1 0 A A A A A A    mem[AAAAAA] = Accu
JCC    1 1 D D D D D D    Set PC to DDDDDD if carry == 0; Always clear carry
```

Synthesising instructions

Pseudo operation	Machine instructions	Comments
CLR	NOR allones	Clear accumulator ('allones' has every bit set to 1)
LDA mem	NOR allones ADD mem	Works by adding the memory value to a zeroed accumulator
LSL	STA tmp ADD tmp	Shift accumulator left & into Carry ('tmp' is a temporary mem variable)
NOT	NOR zero	Invert accumulator ('zero' contains 0)
JMP dest	JCC dest JCC dest	*Always* jump to destination (will also clear carry)
JCS dest	JCC *+2 JCC dest	Jump if carry set (will clear carry) ('*' is the assembler's Inst Ptr)
SUB mem	STA tmp NOR allones NOR mem ADD tmp ADD one	Subtract memory from accumulator In total: Acc = ~mem + Acc + 1 ('one' contains 1)

MCPU [hard] Deficiencies

"Hard" means show-stoppers for "regular" programming

1. No pointer mechanism
   - Might be possible with self modifying code but very cumbersome!!
2. No concept of sub-routines (or a stack)
   - This requires access to the PC – in some way
3. The Carry after an ADD could to be a Zero flag after a NOR
   - MPROZ uses an 'F' bit in this way; both as a Carry and a Zero flag
4. No interrupt capability
   - Pretty much essential for any RTOS, even if only driven by a timer

Note: There's no disrespect here to the original design, it was supposed to be as simple as possible!

MCPU extensions - 1

- Add “indirection” via memory locations
  - Uses an extra instruction bit to switch between absolute / absolute indirect
  - This approach longstanding; cf: Digital Equipment Corp PDP-1 in 1959

```
NOR  0 0 I=0 A A A A A A      Accu = Accu NOR  mem[AAAAAA]
NORI 0 0 I=1 A A A A A A      Accu = Accu NOR [mem[AAAAAA]]
```

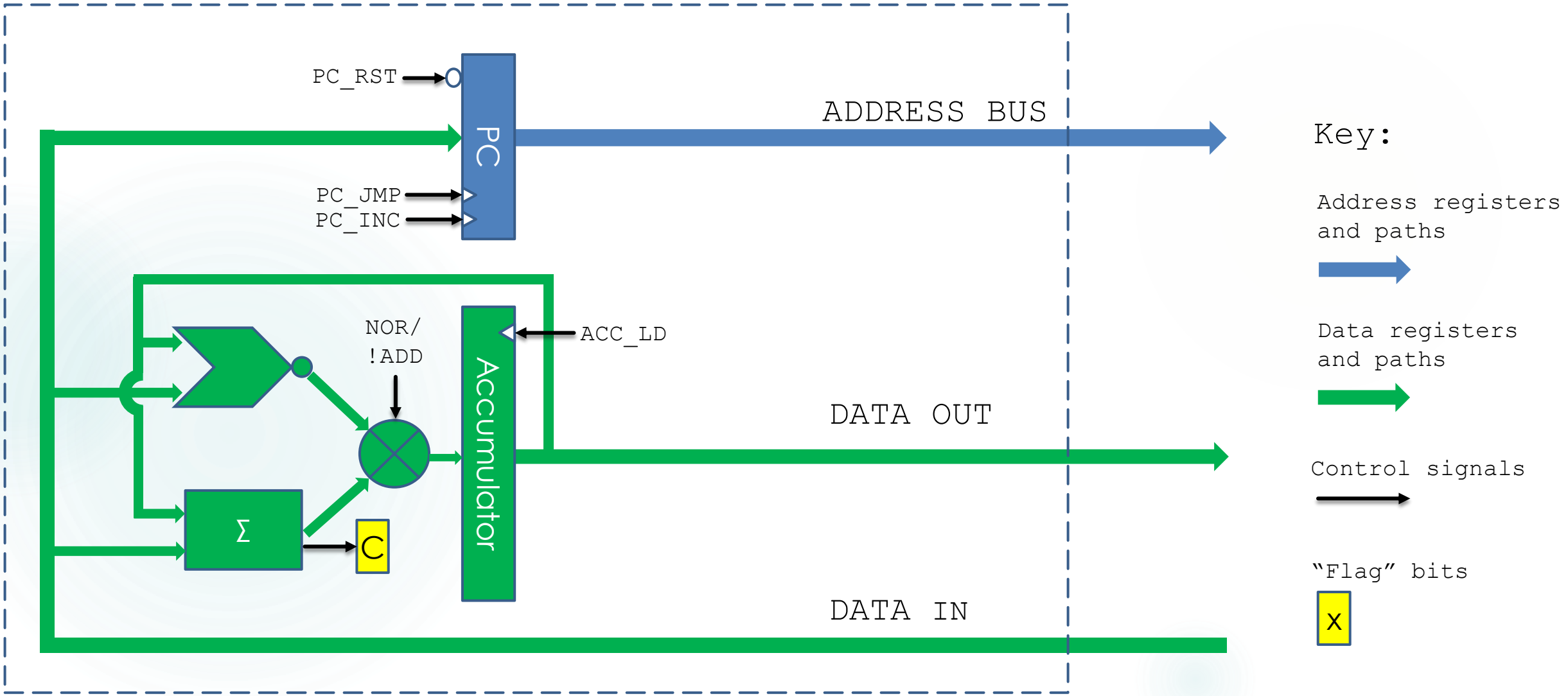
- Needs a “Memory Register” to hold the indirected address
  - Also requires a 2:1 multiplexer to select the PC or MR for the address bus

Widens the instruction portion of the machine word from 2 to 3 bits

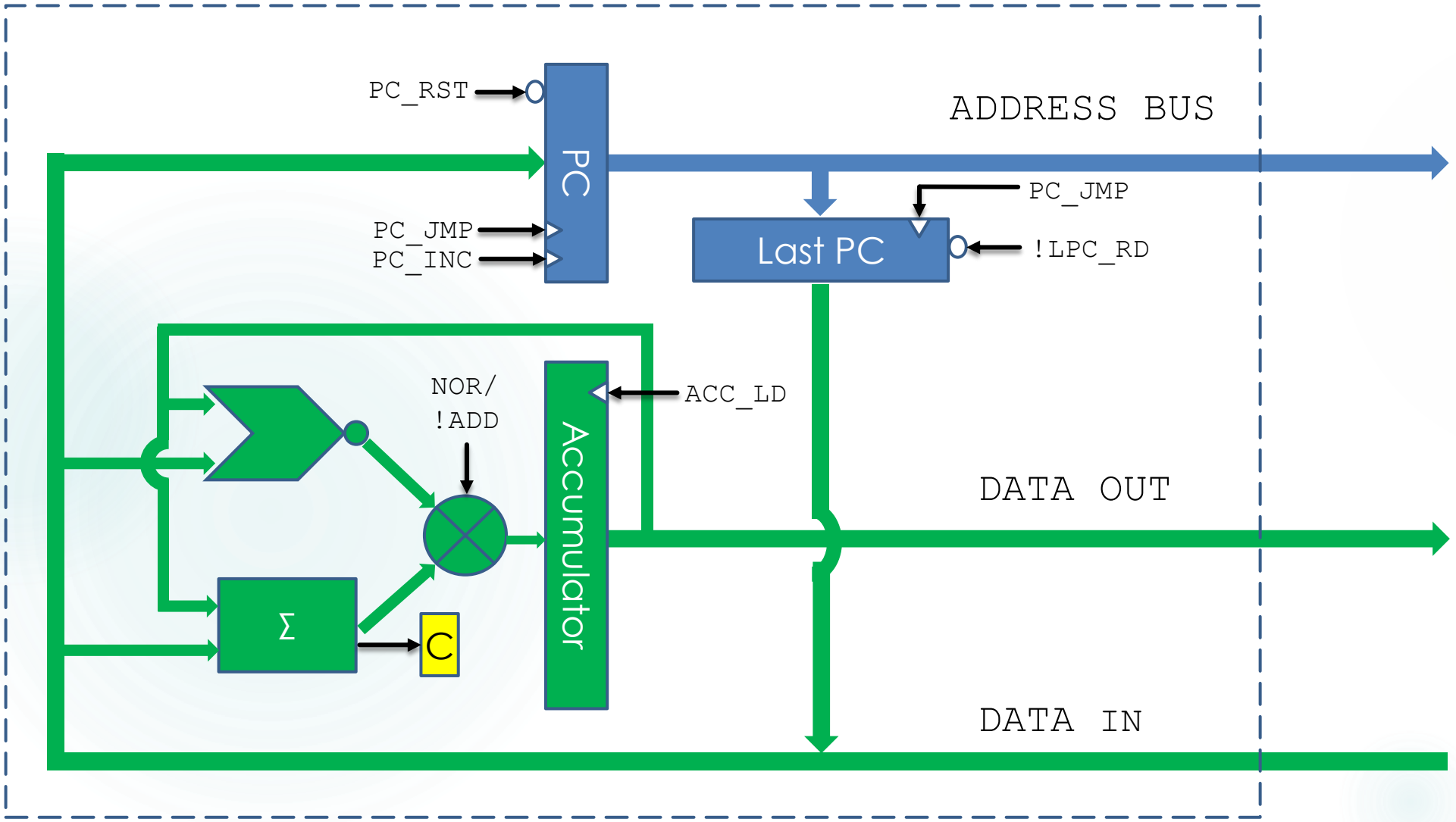
MCPU extensions - 2

- Add a RO “PC pre-jump” register to support subroutines
  - Could equally well be called a “Link Register”
  - Simple hardware but requires some long handed software support
- Add Zero & Negative conditionals (saves a lot of code)
  - Relatively easy combinatorial flag generation
  - Requires more instructions to be decoded to test the additional flags
- Add (single source?) interrupt capability
  - Can be implemented by forcing PC to jump to fixed address
  - Requires ORed requests and interrupt enable/disable controls

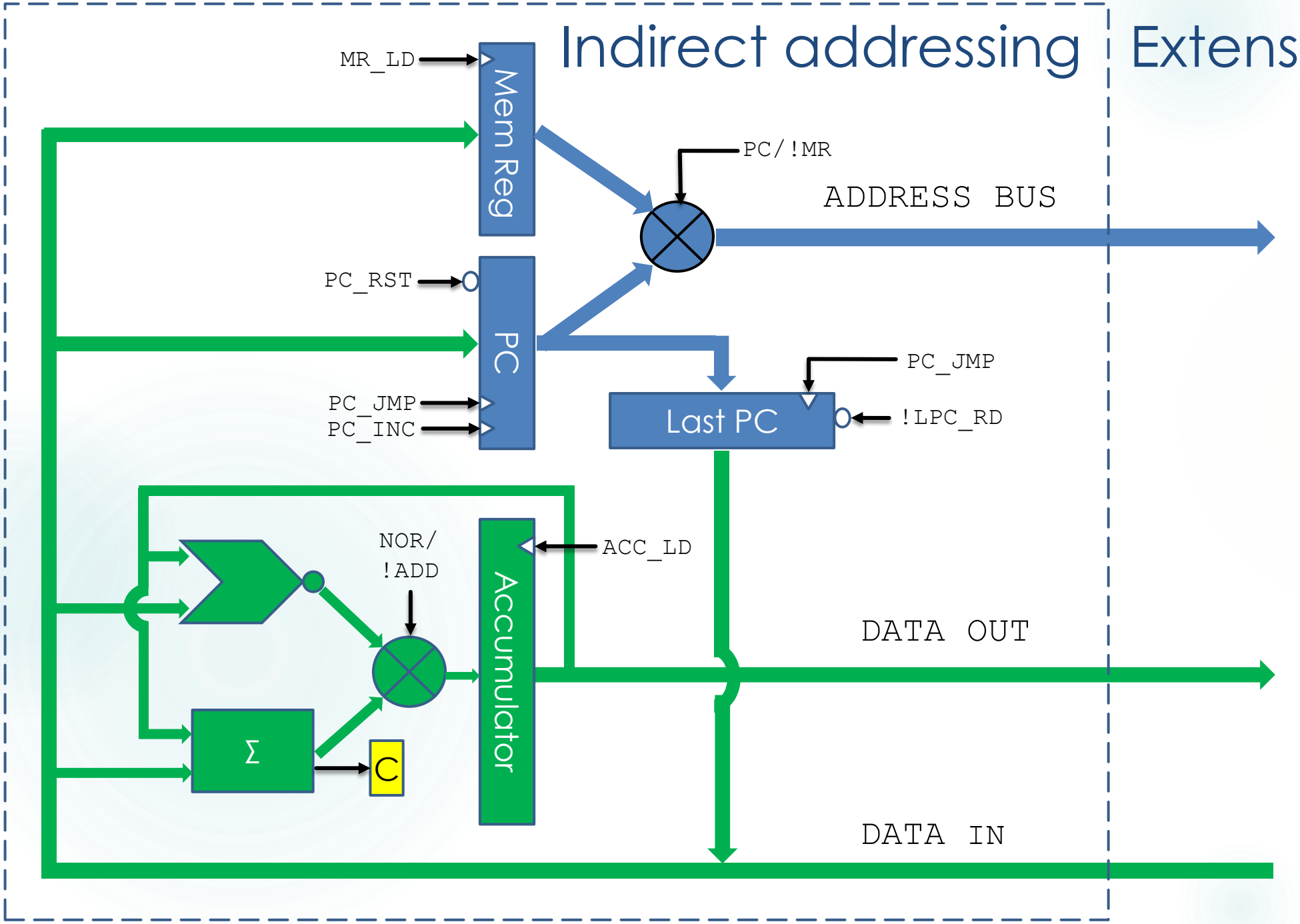
MCPU – Block Diagram

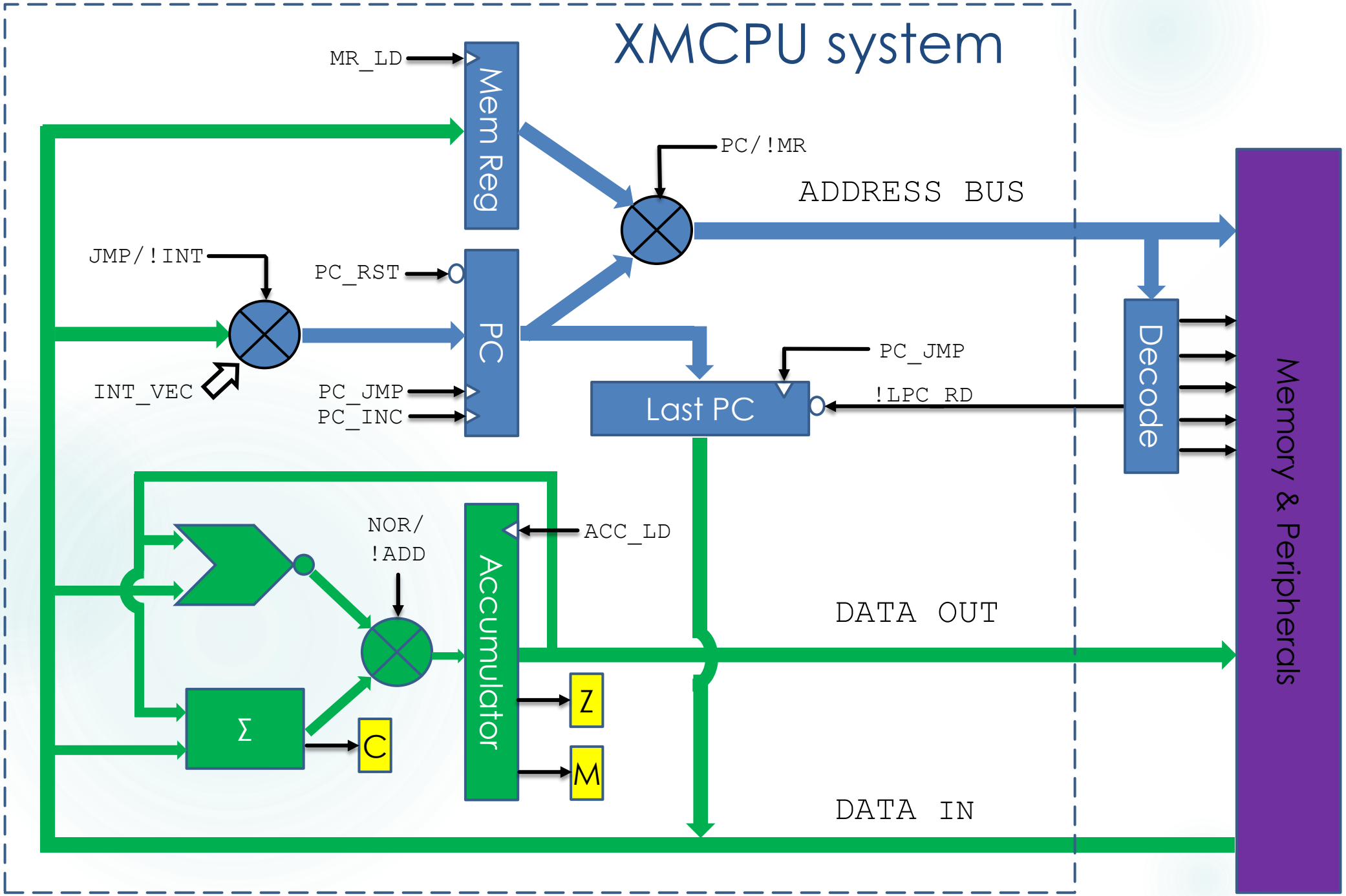


XMCPU – Subroutine extension



Indirect addressing Extension




XMCPU system
MR_LD
Mem Reg
PC/!MR
ADDRESS BUS
JMP/!INT
PC_RST
PC
INT_VEC
PC_JMP
PC_INC
Last PC
PC_JMP
!LPC_RD
Decode
Memory & Peripherals
NOR/
!ADD
ACC_LD
Accumulator
DATA OUT
Σ
C
Z
M
DATA IN

XMCPU – Possible instruction set 1

```
NOR    0 0 0 0 A A A A A A    Accu = Accu NOR mem[AAAAAA]
ADD    0 0 1 0 A A A A A A    Accu = Accu  +  mem[AAAAAA]
STA    0 1 0 0 A A A A A A    mem[AAAAAA] = Accu
JMP    0 1 1 0 D D D D D D    Set PC to DDDDDD

NORI   0 0 0 1 A A A A A A    Accu = Accu NOR [mem[AAAAAA]]
ADDI   0 0 1 1 A A A A A A    Accu = Accu  +  [mem[AAAAAA]]
STAI   0 1 0 1 A A A A A A    mem[AAAAAA] = Accu
JMPI   0 1 1 1 D D D D D D    Set PC to [DDDDDD]

JEQ    1 0 0 0 D D D D D D    Set PC to DDDDDD if Zero  == 1
JNE    1 0 1 0 D D D D D D    Set PC to DDDDDD if Zero  != 1
JCS    1 1 0 0 D D D D D D    Set PC to DDDDDD if Carry == 1
JMI    1 1 1 0 D D D D D D    Set PC to DDDDDD if Minus == 1

JEQI   1 0 0 1 D D D D D D    Set PC to [DDDDDD] if Zero  == 1
JNEI   1 0 1 1 D D D D D D    Set PC to [DDDDDD] if Zero  != 1
JCSI   1 1 0 1 D D D D D D    Set PC to [DDDDDD] if Carry == 1
JMII   1 1 1 1 D D D D D D    Set PC to [DDDDDD] if Minus == 1
```

XMCPU - Possible instruction set 2

```
NOR    0 0 0 0 A A A A A A     Accu = Accu NOR mem[AAAAAA]
ADD    0 0 1 0 A A A A A A     Accu = Accu  +  mem[AAAAAA]
STA    0 1 0 0 A A A A A A     mem[AAAAAA] = Accu
JMP    0 1 1 0 D D D D D D     Set PC to DDDDDD

NORI   0 0 0 1 A A A A A A     Accu = Accu NOR [mem[AAAAAA]]
ADDI   0 0 1 1 A A A A A A     Accu = Accu  +  [mem[AAAAAA]]
STAI   0 1 0 1 A A A A A A     mem[AAAAAA] = Accu
JMPI   0 1 1 1 D D D D D D     Set PC to [DDDDDD]

JEQ    1 0 0 0 D D D D D D     Set PC to DDDDDD if Zero  == 1
JNE    1 0 1 0 D D D D D D     Set PC to DDDDDD if Zero  != 1
JCS    1 1 0 0 D D D D D D     Set PC to DDDDDD if Carry == 1
JMI    1 1 1 0 D D D D D D     Set PC to DDDDDD if Minus == 1
Add "Immediate" instructions using sign extension
NORIM  1 0 0 1 D D D D D D     Accu = Accu NOR sign_extend(DDDDDD)
ADDIM  1 0 1 1 D D D D D D     Accu = Accu  +  sign_extend(DDDDDD)
LDIM   1 1 0 1 D D D D D D     Accu = sign_extend(DDDDDD)
INH    1 1 1 1 x x x x x x     Inherent: eg: SEC, CLC, SEI, CLI, HALT, etc.
```

An example ‘complex’ coding

- PUSH the accumulator (SP is simply a dedicated memory location)

```
3       STAI SP             ; write to stack location via [SP]
2       STA tmp             ; save accumulator
2+2     LDA SP              ; load Stack Pointer value
2       ADD allones         ; subtract 1
2       STA SP              ; update Stack Pointer
2+2     LDA tmp             ; recover original accumulator value
```

Which takes 17 machine cycles

Project XMCPU

- A TTL implementation of an extended MCPU design
  - Where "TTL" is most likely 74**HC**xx
    - PC, accumulator: 74161 4-bit synchronous counters with load and reset
    - ALU: 74283 4 bit adders with carry, NOR using 7402 4 * 2 i/p NOR gates
    - 2:1 Muxes: 74157, 4 * 2 to 1 multiplexor (or 4:1 Muxes: 74153, 2 * 4 to 1)
- 24 bit data/instruction width
  - Containing 4 or 5 instruction bits: 1M or ½ M word memory space
- Expected clock rate is 8 to 10 MHz
  - ADD instructions need an extra cycle for carry propagation
- This will be build one 2-layer PCB[s] using DIL devices
  - CAD done using KICad (there is too much wiring to do by hand)

XMCPU Software

- The original project used SMAL32

SMAL32 is a symbolic macro assembly language for 32 bit computers. The assembler has no built-in knowledge of any machine's instruction set.

- XMCPU software will written using [at least] a macro assembler
  - Use of macros allows relatively complex forms to be built
  - FORTH likely to be practicable
  - **Tiny** C compilers *might* be retargetable, eg: otcc and cc500

The ultimate goal is a "self-hosted" toolchain, *not* PC built object running on the system

XMCPU System peripherals

- Two 8-N-1 UARTs, interrupt generating
  1. Console connection
     - Also capable of *multi* "virtual terminal" via simple wrapper protocol
  2. Network interface providing TCP/IP connectivity over SLIP
- Free running, Data Bus width, RO, microsecond rate counter
  - Useful for timestamping and accurate non/blocking delays
- Interrupt generating, DB width, WO, "counter match register"
  - Pretty much essential for any RTOS!
- 24 switches to enter bootstrap(also accessible via RO register)

And many, many LEDs, of course :)

The end

ANY QUESTIONS?