Enrico Nardelli
Logic Circuits and Computer Architecture

Appendix B
The design of VS0: a very simple CPU

Instruction set

- Just 4 instructions

LOAD M - Copy into Accumulator the value from memory at address M

STORE M - Save Accumulator value into memory at address M

ADD M - Sum values of Accumulator and of memory at address M and put the result into the Accumulator

JUMP A - Execute in the next step the instruction stored at address A of memory

Registers and Memory

- The bare minimum

PC - Program Counter

IR - Instruction Register

MAR - Memory Address Register

MBR - Memory Buffer Register

AC - Accumulator

- All registers have 8 bits
- 64 (2^6) bytes of memory, each with 8 bits

Instruction format

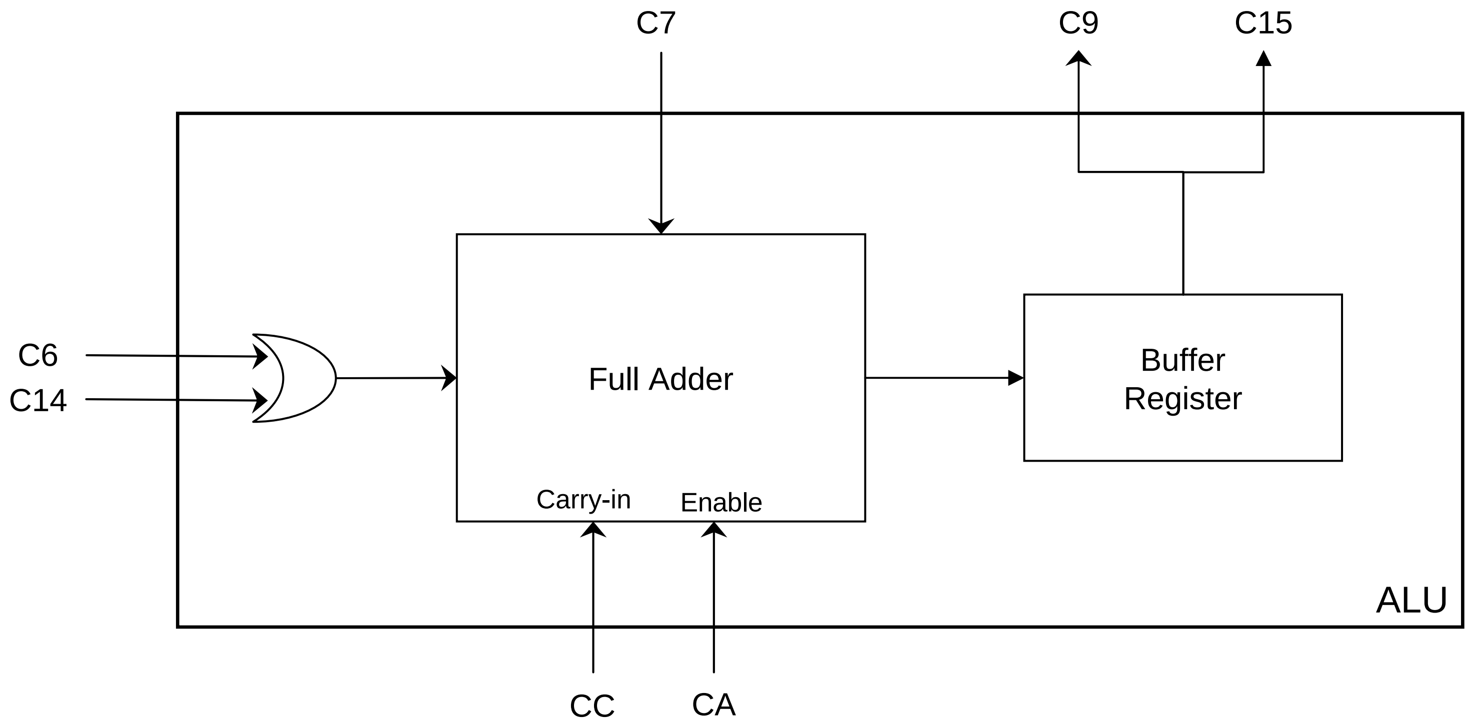
- 2 bits for the opcode
- 6 bits for the address (b_5 is the MSB, b_0 is the LSB)

LOAD	0 0	$b_5\ b_4\ b_3\ b_2\ b_1\ b_0$
STORE	1 1	$b_5\ b_4\ b_3\ b_2\ b_1\ b_0$
ADD	0 1	$b_5\ b_4\ b_3\ b_2\ b_1\ b_0$
JUMP	1 0	$b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

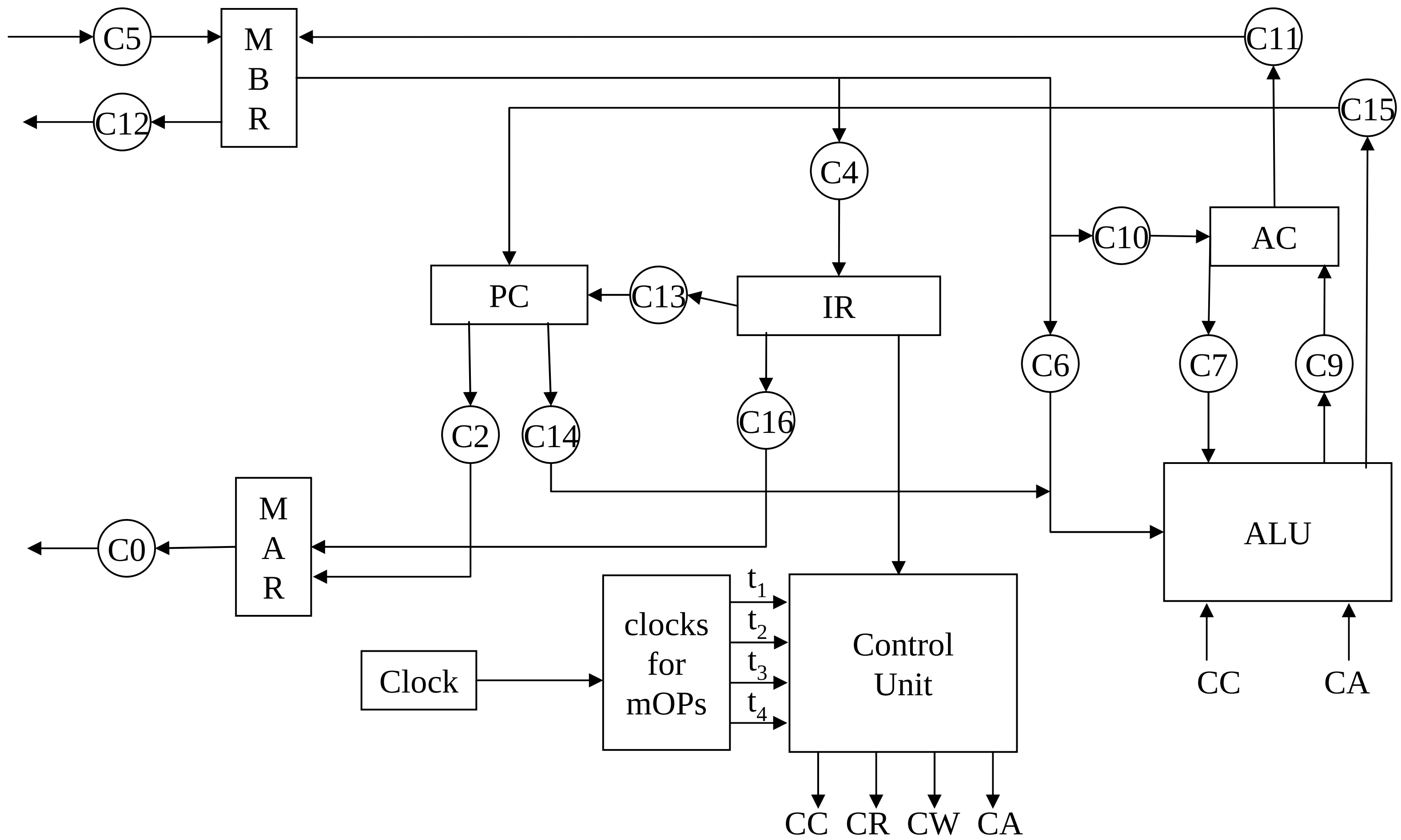
ALU’s organization

- Only capable of adding (signal CA) two 8 bits number with a possible carry-in (signal CC)
- No overflow signal
- One addend is the Accumulator
- The other addend is the selection between PC and a memory address (through C6 and C14)
- ALU’s output is stored into an internal buffer register

ALU's internal structure



Internal schema



Micro-operations (1)

- Fetch

t1:	MAR <- PC		C2				
t2:	MBR <- (memory); (PC)+1;						
		C0	C5	C14	CA	CC	CR
t3:	PC <- (ALU); IR <- (MBR)						
			C4	C15			

- Execute ADD

t1:	MAR <- (IR_{addr})	C16		
t2:	MBR <- (memory)	C0	C5	CR
t3:	(AC)+(MBR)	C6	C7	CA
t4:	AC <- (ALU)	C9		

Micro-operations (2)

- Execute LOAD

t1:	MAR <- (IR_{addr})	C16		
t2:	MBR <- (memory)	C0	C5	CR
t3:	AC <- (MBR)	C10		

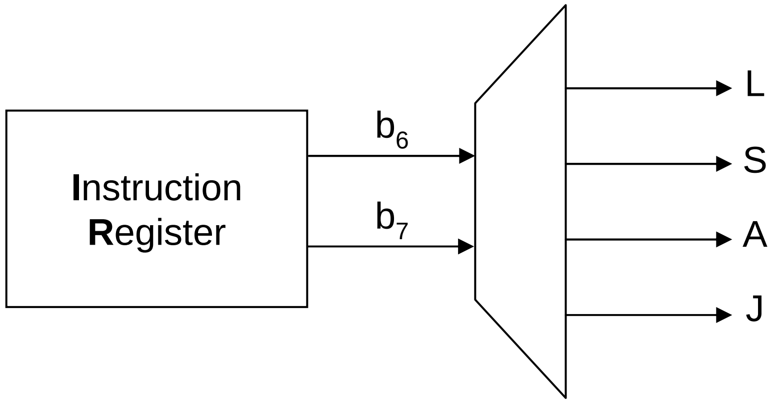
- Execute STORE

t1:	MAR <- (IR_{addr}); MBR <- (AC)			
		C11	C16	
t2:	memory <- (MBR)	C0	C12	CW

- Execute JUMP

t1:	PC <- (IR_{addr})	C13

Decoding instructions

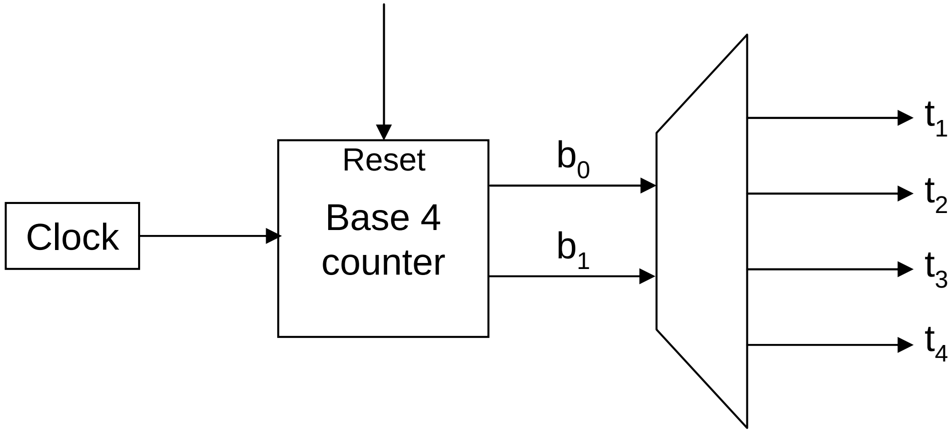


- Inside the Control Unit a 2-to-4 decoder provides L, S, A, J signals denoting which instruction is currently in IR

Micro-operations (3)

- Generate t_1,t_2,t_3,t_4 from the clock through a base-4 counter and a 2-to-4 decoder
- Distinguish between Fetch and Execute with a 1-bit state register (can be inside the Control Unit)
- For each control signal Cn write the boolean expression for its activation in terms of status (Fetch/Execute), mOP step being executed (t_1,t_2,t_3,t_4), and operation to be executed (L,S,A,J), by scanning the list of activated control signals for each step of each mOP

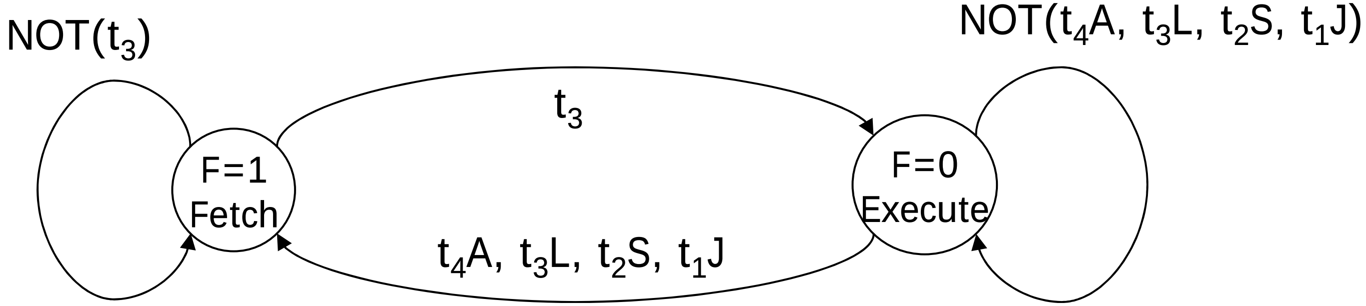
Generating clocks for mOPs



- Counter can be reset to optimize at the last step of each mOP
- Reset signal: $Ft_3 + F't_1J + F't_2S + F't_3L$

State representation

- Control unit can be in the state of fetch (F=1) or in the state of execute (F=0)
- Status changes are activated during the last mOP step of each phase of fetch or execute



- There is just one boolean expression for the transition condition of the unique state variable

$$F_{n+1} = F_n t_3' + F_n' t_4 A + F_n' t_3 L + F_n' t_2 S + F_n' t_1 J$$

Activation of control signals

Control Signal	Boolean expression	Control Signal	Boolean expression
C0	$Ft_2 + F't_2A + F't_2L + F't_2S$	C12	$F't_2S$
C2	Ft_1	C13	$F't_1J$
C4	Ft_3	C14	Ft_2
C5	$Ft_2 + F't_2A + F't_2L$	C15	Ft_3
C6	$F't_3A$	C16	$F't_1A + F't_1L + F't_1S$
C7	$F't_3A$	CC	Ft_2
C9	$F't_4A$	CA	$Ft_2 + F't_3A$
C10	Ft_3L	CR	$Ft_2 + F't_2A + F't_2L$
C11	$Ft_3 + F't_1S$	CW	$F't_2S$

A note on boolean expressions (1)

- Boolean expressions written have been derived directly from inspection of mOPs
- The theory of circuit synthesis tells us to examine what happens in general to each output signal for each possible combination of input signals (t_1, t_2, t_3, t_4, L, S, A, J) and state signal (F)
- Writing, e.g., $F' t_2 L$ could be wrong, since the exact and complete term is $F' t_1' t_2 t_3' t_4' L S' A' J'$: this is not equivalent to the former, which corresponds to $F'(t_1+t_1')t_2(t_3+t_3')(t_4+t_4')L(S+S')(A+A')(J+J')$

A note on boolean expressions (2)

- But we know that among t_1, t_2, t_3, and t_4 only and exactly one can be true, therefore we can substitute, e.g., $t_1' t_2$ with $(t_1+t_1')t_2$ knowing that the condition $t_1 t_2$ can never be true and hence derive the correct simpler term t_2 (in other words, $t_1 t_2$ is a **don’t care** condition)
- For signals L, S, A, and J, if one of them is true then all the others are false and the same reasoning above applies.
- Finally, there are also those situations (e.g., for C2 activation in mOP t_1 during the fetch phase) where we **don’t care** at all about which of these signals is true

Global optimization of signals

Signal	Boolean expression	Signal	Boolean expression
CC, C14	Ft_2		
C2	Ft_1	C13	$F' t_1 J$
C4, C15	Ft_3	CW, C12	$F' t_2 S$
C5, CR	$CC + F' t_2 A + F' t_2 L$	C16	$F' t_1 A + F' t_1 L + F' t_1 S$
C6, C7	$F' t_3 A$	C0	C5 + CW
		CA	CC + C6
C9	$F' t_4 A$		
C10	$F' t_3 L$		
C11	$C4 + F' t_1 S$	F_{n+1}	$Ft_3' + C9 + C10 + CW + C13$

Additional considerations

- Do we need both state signal and instruction signals L,S,A,J to activate control signals?
  - e.g. in the activation expression for C7, instead of F' t_3A, can we just write t_3A ?
    - no, because if the previously fetched instruction was also an ADD then C7 is (wrongly) activated also during mOP step t_3 in the fetch phase
  - hence we need both state signal and instruction signals
- Do we need an explicit representation for state ?
  - no, if we use for the execution phases a different set of clock signals t_4,t_5,t_6,t_7
  - What changes using this approach? What do we lose?

No Explicit State: Micro-operations (1)

- Fetch

t1: MAR <- PC C2

t2: MBR <- (memory); (PC)+1;
C0 C5 C14 CA CC CR

t3: PC <- (ALU); IR <- (MBR)
C4 C15

- Execute ADD

t4: MAR <- (IR_{addr}) C16

t5: MBR <- (memory) C0 C5 CR

t6: (AC)+(MBR) C6 C7 CA

t7: AC <- (ALU) C9

No Explicit State: Micro-operations (2)

- Execute LOAD

t4:	MAR <- (IR_{addr})	C16		
t5:	MBR <- (memory)	C0	C5	CR
t6:	AC <- (MBR)	C10		

- Execute STORE

t4:	MAR <- (IR_{addr}); MBR <- (AC)			
		C11	C16	
t5:	memory <- (MBR)	C0	C12	CW

- Execute JUMP

t4:	PC <- (IR_{addr})	C13

No Explicit State: Micro-operations (3)

- Generate $t_1,t_2,t_3,t_4,t_5,t_6,t_7$ from the clock through a base-8 counter and a 3-to-8 decoder (possibly use a counter with reset for optimization)
- For each control signal Cn write the boolean expression for its activation in terms of mOP step being executed ($t_1,t_2,t_3,t_4,t_5,t_6,t_7$), and operation to be executed (L,S,A,J), by scanning the list of activated control signals for each step of each mOP

No Explicit State: Activation of control signals

Control Signal	Boolean expression
C0	$t_2 + t_5A + t_5L + t_5S$
C2	t_1
C4	t_3
C5	$t_2 + t_5A + t_5L$
C6	t_6A
C7	t_6A
C9	t_7A
C10	t_3L
C11	$t_3 + t_4S$

Control Signal	Boolean expression
C12	t_5S
C13	t_4J
C14	t_2
C15	t_3
C16	$t_4A + t_4L + t_4S$
CC	t_2
CA	$t_2 + t_6A$
CR	$t_2 + t_5A + t_5L$
CW	t_5S

The complete circuit

- All circuital elements (including the Control Unit) have now been defined and it is known how to realize them
- Try drawing the complete circuit for the CPU and the memory!!
- It is a long but worthwhile task
- Do it in hierarchical stages: first layout modules and afterwards layout gates within modules
- In the real life they use CAD systems for electronic circuit design !

A trivial program

- Give at location SUM the sum of four numbers stored in locations of memory N1, N2, N3, N4

```
; Location SUM is distinct from N1, N2, N3, N4
LOAD N1          ;  AC <- N1
ADD N2           ;  AC <- N1+N2
ADD N3           ;  AC <- N1+N2+N3
ADD N4           ;  AC <- N1+N2+N3+N4
STORE SUM        ;  SUM <- N1+N2+N3+N4
```

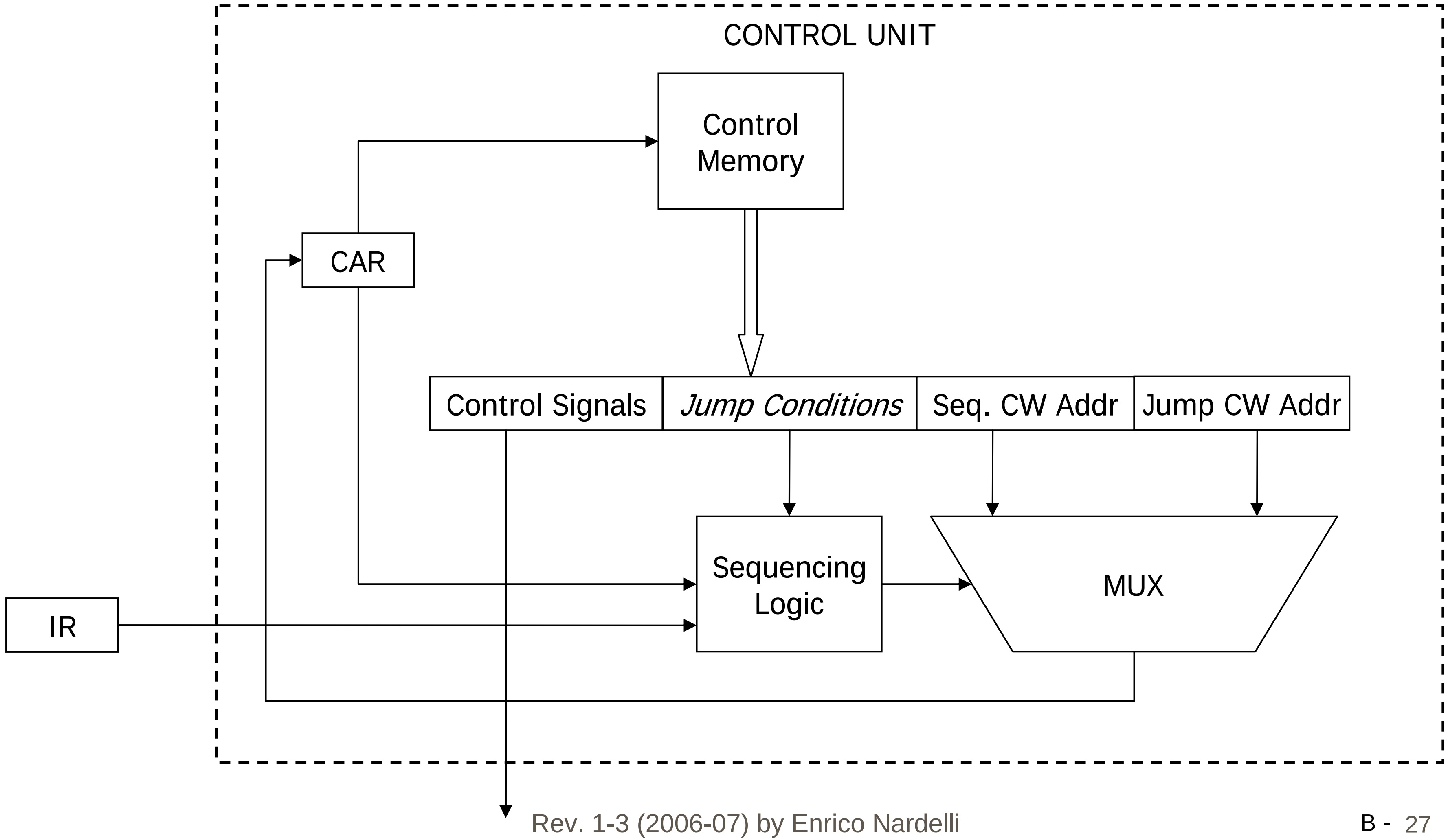
Control Unit's implementation with micro-programmed control

- For the implementation of CU with a micro-programmed approach we **do not need**:
  - signals t_1 ... t_n marking different mOPs
  - state register distinguishing between fetch and execute
- Even the IR decoder is not really needed, but we may use it depending on the CW structure
- Structure of CW and structure of Sequencing Logic are strictly related: a CW with more information needs a simpler Sequencing Logic and vice-versa

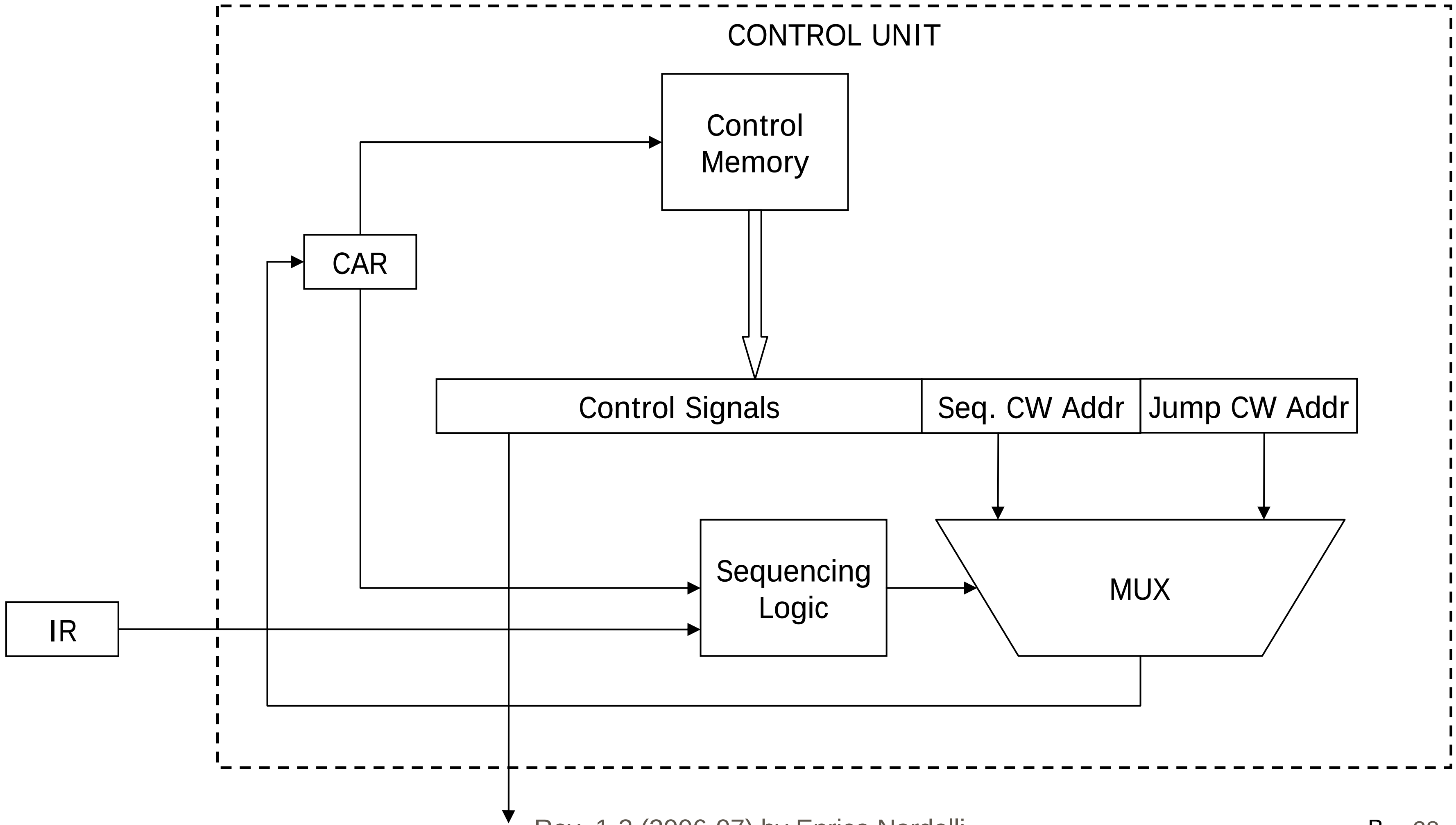
CW and sequencing mOPs

- CW has two address fields (SmA and JmA) of 5 bits each
  - SmA is the next CW address in case of sequential execution
  - JmA is the next CW address in case of jump
  - Fields are empty when the choice is forced
- A 2-way multiplexer is used to select between SmA and JmA and hence choose the next CW to be executed
- Selection line (SEL) for multiplexer is activated by a circuit in the Sequential Logic whose structure depends on the structure of jump conditions in CW
  - No jump flags
  - One jump flag (K) only for end-of-mOP
  - Jump flags both for end-of-mOP and for selecting the proper micro-procedure during the CPU execution phase

Generic structure of CU



CW without jump conditions: CU's structure



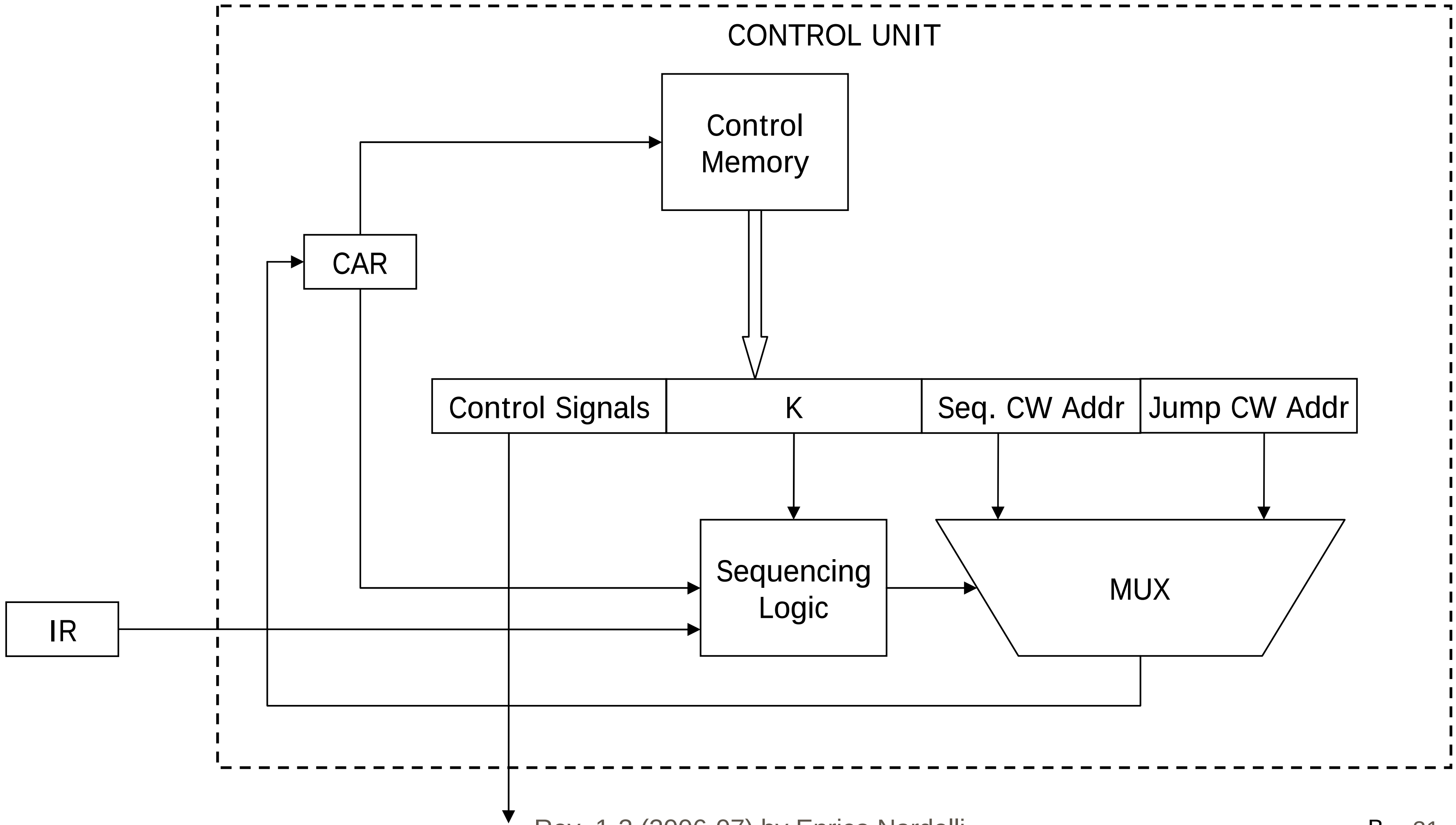
CW without jump conditions: Sequencing Logic

- If there are no flags in CW the selection between SmA and JmA may use only the state of CU, represented by CAR value
- Towards the end of CU execution cycle, CAR contains the address of current CW in execution hence the value of such an address is used to drive the selection of next CW
- A CAR decoder provides I_n signals telling that CW at address *n* is being executed
- A 2-to-4 decoder on the two most significant bits of IR is needed to understand which CPU instruction is being executed and to provide L, S, A, and J signals
- Signal for selection line (0 to select SmA, 1 for JmA) is
  - $SEL = I_3 + I_6 + I_{12} + I_{16} + I_{17} + I_7L + I_8S + I_9A + I_{10}J$
- Both decoders are part of the Sequencing Logic

CW without jump conditions: Control Memory

Micro Procedure	mA	C0	C2	C4	C5	C6	C7	C9	C 10	C 11	C 12	C 13	C 14	C 15	C 16	CC	CA	CR	CW	S mA	J mA
Fetch	1		1																	2	
	2	1			1								1			1	1	1		3	
	3			1										1							7
Load	4														1					5	
	5	1			1													1		6	
	6								1												1
Execute	7																			8	4
	8																			9	11
	9																			10	13
	10																			7	17
Store	11										1				1					12	
	12	1									1								1		1
Add	13														1					14	
	14	1			1													1		15	
	15					1	1										1			16	
	16							1													1
Jump	17											1									1

CW with one jump condition: CU's structure



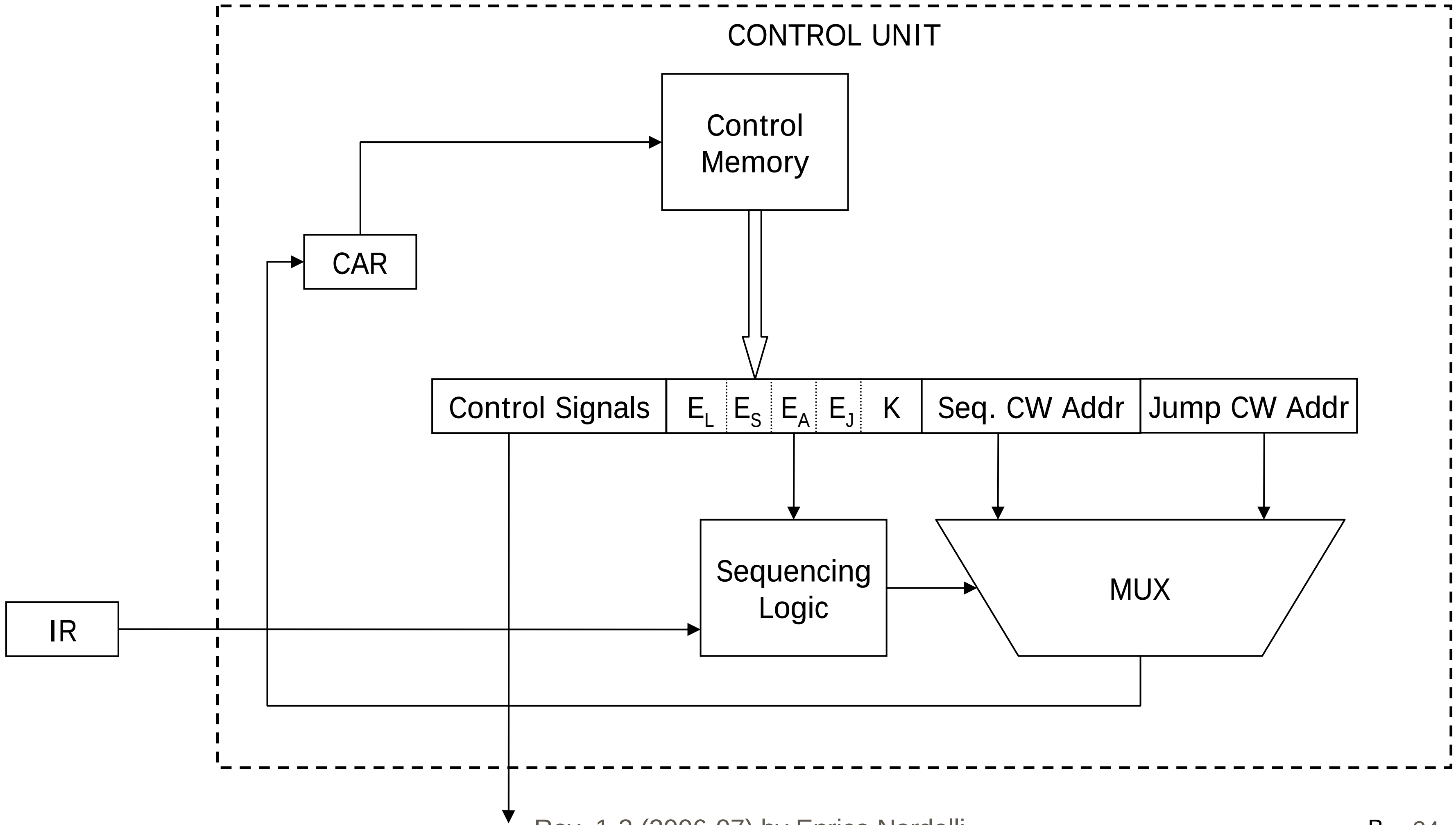
CW with one jump condition: Sequencing Logic

- A jump flag (K) is used to mark the last mOP of each micro-procedure (but for the Execute one)
- I_n signals provided by the CAR decoder now are only needed during the Execute micro-procedure
- A 2-to-4 decoder on the two most significant bits of IR is needed to understand which CPU instruction is being executed and to provide L, S, A, and J signals
- Signal for selection line (0 to select SmA, 1 for JmA) is
  - $SEL = K + I_7L + I_8S + I_9A + I_{10}J$
- Sequencing Logic is independent from the location of any micro-procedure in Control Memory, but for the Execute one

CW with one jump condition: Control Memory

Micro Procedure	mA	C0	C2	C4	C5	C6	C7	C9	C 10	C 11	C 12	C 13	C 14	C 15	C 16	CC	CA	CR	CW	K	S mA	J mA
Fetch	1		1																		2	
	2	1			1								1			1	1	1			3	
	3			1										1						1		7
Load	4														1						5	
	5	1			1													1			6	
	6								1											1		1
Execute	7																				8	4
	8																				9	11
	9																				10	13
	10																				7	17
Store	11										1				1						12	
	12	1									1								1	1		1
Add	13														1						14	
	14	1			1													1			15	
	15					1	1										1				16	
	16							1												1		1
Jump	17											1								1		1

CW with many jump conditions: CU's structure



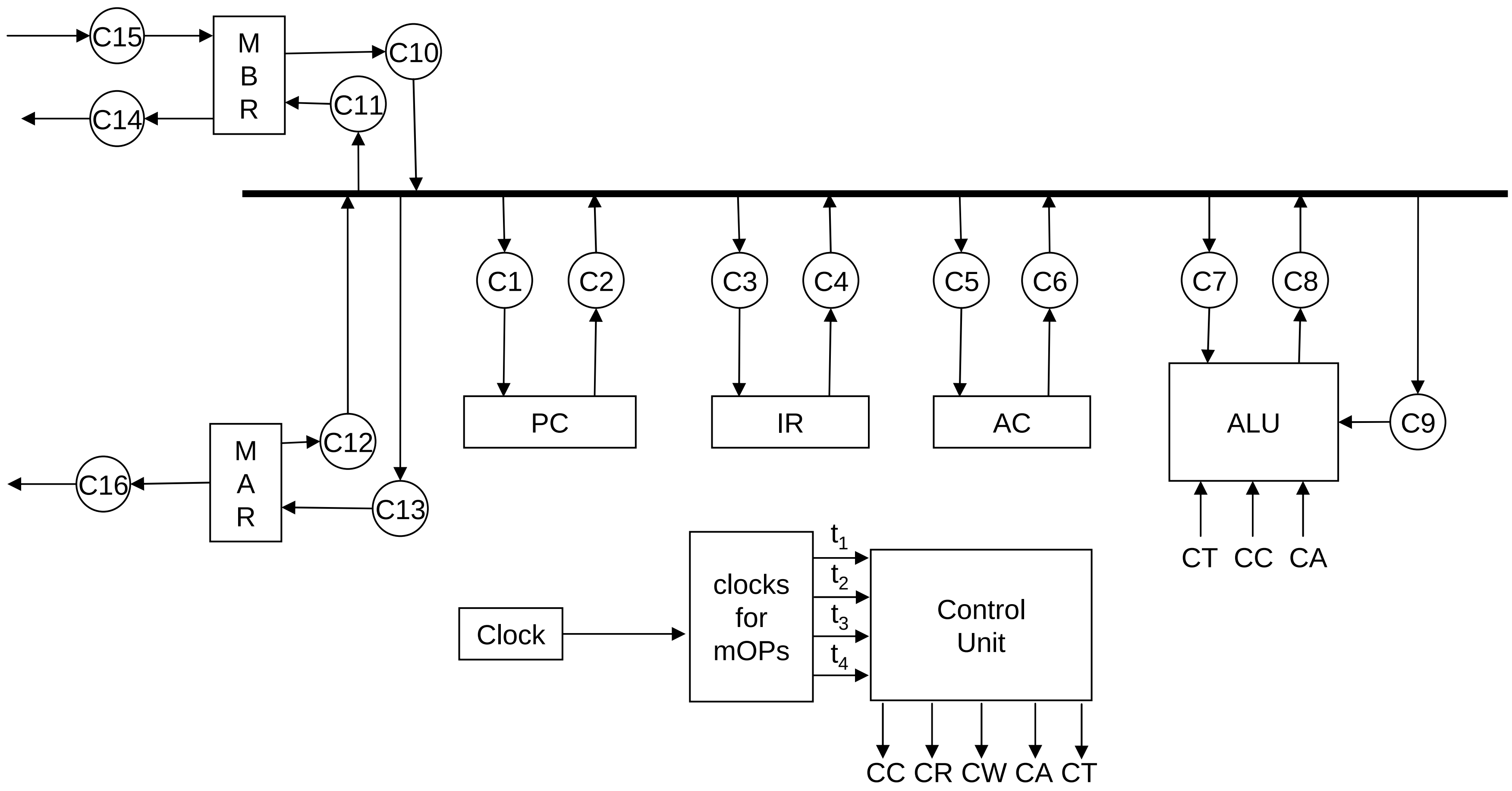
CW with many jump conditions: Sequencing logic

- A jump flag (K) is used to mark the last mOP of each micro-procedure
- Four jump flags (E_L, E_S, E_A, E_J) mark the four mOPs in the Execute micro-procedure
- There is no need now for a CAR decoder: this is obtained at the cost of a longer CW
- A 2-to-4 decoder on the two most significant bits of IR is needed to understand which CPU instruction is being executed and to provide L, S, A, and J signals
- Signal for selection line (0 to select SmA, 1 for JmA) is
  - $SEL = K + E_L L + E_S S + E_A A + E_J J$
- Sequencing Logic is now fully independent from the location of any micro-procedure in Control Memory

CW with many jump conditions: Control Memory

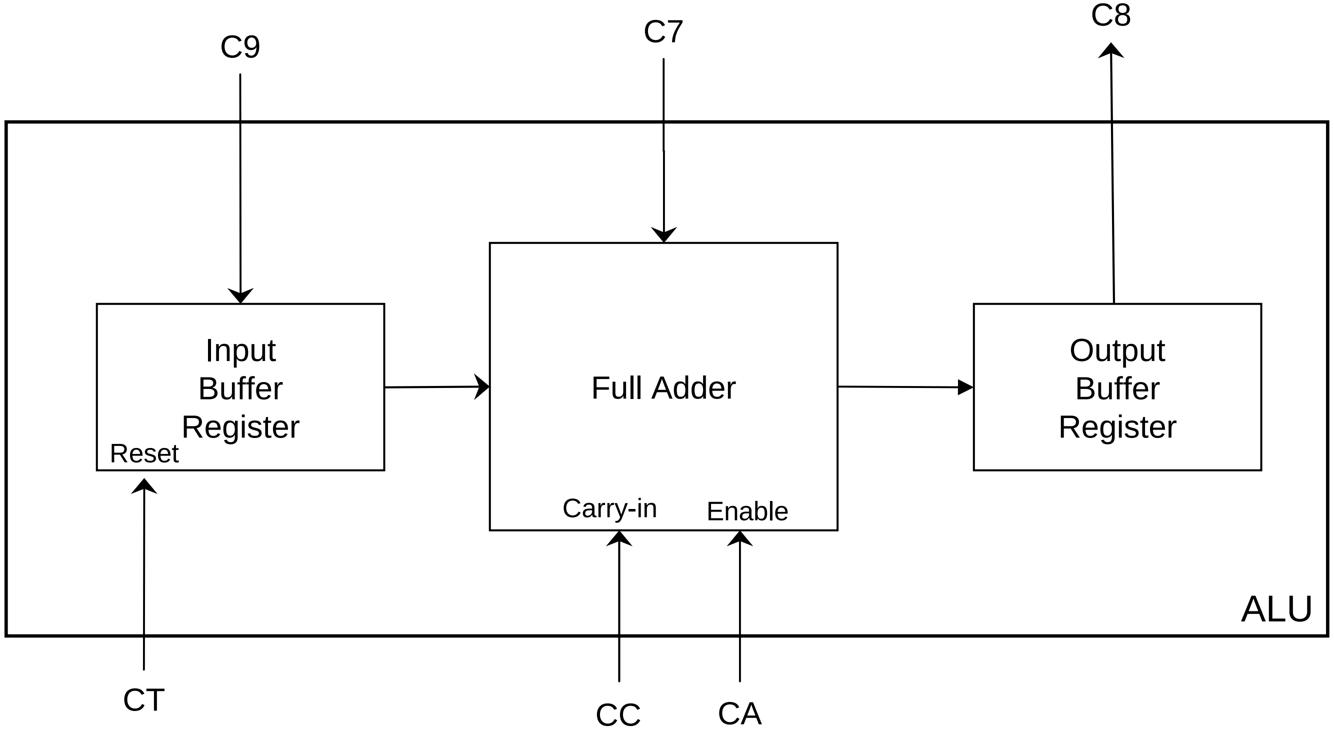
Micro Procedure	mA	C0	C2	C4	C5	C6	C7	C9	C 10	C 11	C 12	C 13	C 14	C 15	C 16	CC	CA	CR	CW	EL	ES	EA	EJ	K	S mA	J mA
Fetch	1		1																						2	
	2	1			1								1			1	1	1							3	
	3			1										1										1		7
Load	4														1										5	
	5	1			1													1							6	
	6								1															1		1
Execute	7																			1					8	4
	8																				1				9	11
	9																					1			10	13
	10																						1		7	17
Store	11										1				1										12	
	12	1									1								1					1		1
Add	13														1										14	
	14	1			1													1							15	
	15					1	1										1								16	
	16							1																1	17	1
Jump	17											1												1		1

An internal schema with single bus



ALU changes

- ALU needs a buffer (with reset) also for input



Micro-operations (1) Single Bus

- Fetch
  - one more step

t1:	MAR <- PC		C2	C13		
t2:	MBR <- (memory)		C16	C15	CR	
	(PC)+1	C2	C7	CT	CA	CC
t3:	PC <- (ALU)		C8	C1		
t4:	IR <- (MBR)		C10	C3		

Micro-operations (2) Single Bus

- Execute ADD
  - reorganization of micro-operations

t1:	MAR <- (IR_{addr})	C4	C13	
t2:	MBR <- (memory)	C16	C15	CR
	ALU <- (AC)	C6	C9	
t3:	(MBR)+(ALU)	C10	C7	CA
t4:	AC <- (ALU)	C8	C5	

Micro-operations (3) Single Bus

- Execute LOAD

t1:	MAR <- (IR_{addr})	C4	C13	
t2:	MBR <- (memory)	C16	C15	CR
t3:	AC <- (MBR)	C10	C5	

Micro-operations (4) Single Bus

- Execute STORE
  - one more step

t1:	MAR <- (IR_{addr})	C4	C13	
t1:	MBR <- (AC)	C6	C11	
t2:	memory <- (MBR)	C14	C16	CW

- Execute JUMP

t1:	PC <- (IR_{addr})	C14	C2

Completion of single bus

- Continue development as shown before
- Decide whether to explicitly represent state or not
- Decide whether to implement a hardwired CU or a micro-programmed one
- In the latter case, decide the structure of the control word

Other simple design variations

- **Try them** (even together) to understand consequences of various design decisions !
  1. Add to the ALU the capability to provide Zero or Overflow signal and use a JUMP conditional to the signal value instead of the unconditional JUMP
  2. Use an internal CPU schema with two internal buses to connect CPU elements instead of direct paths
  3. Use two variants of ADD. One, specified by $b_5=0$, having as parameter the address of memory cell, written in the byte right after the one with ADD. The other, specified by $b_5=1$, having as argument the number to be added written in bits b_4-b_0
  4. Use a micro-programmed CU with just one address field
  5. Study if it is possible to avoid the use of the 2-to-4 IR decoder by means of a different organization of the micro-procedure for the CPU execution phase