Petite Histoire des Ordinateurs Pédagogiques

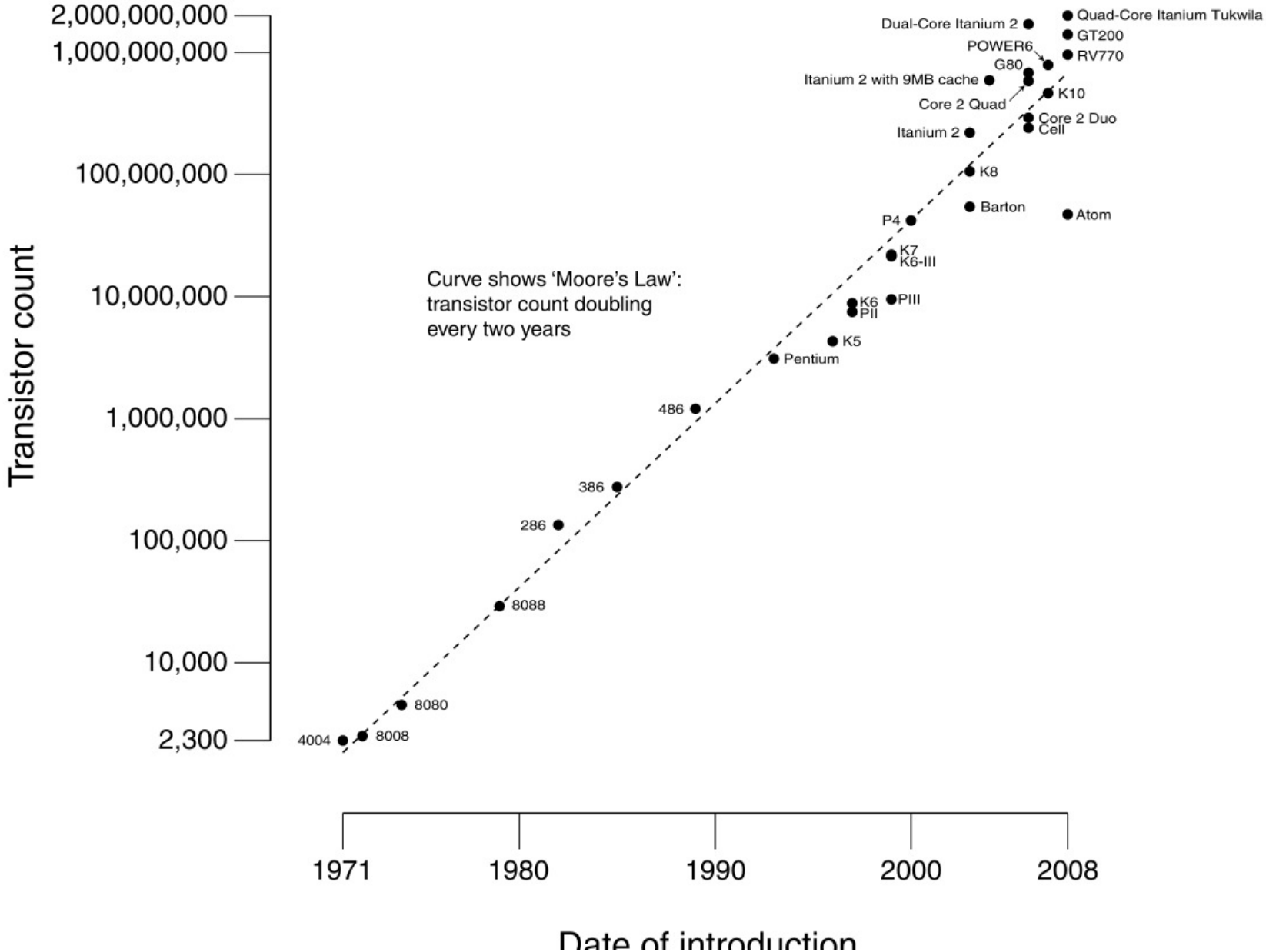
Alain Guyot
http://users-tima.imag.fr/cis/guyot/Cours/
http://www.aconit.org/histoire/calcul_mecanique/

Beaucoup d'enseignant en architecture des ordinateurs on eu la tentation de développer des outils logiciels pour évaluer l'impact de décisions architecturales.

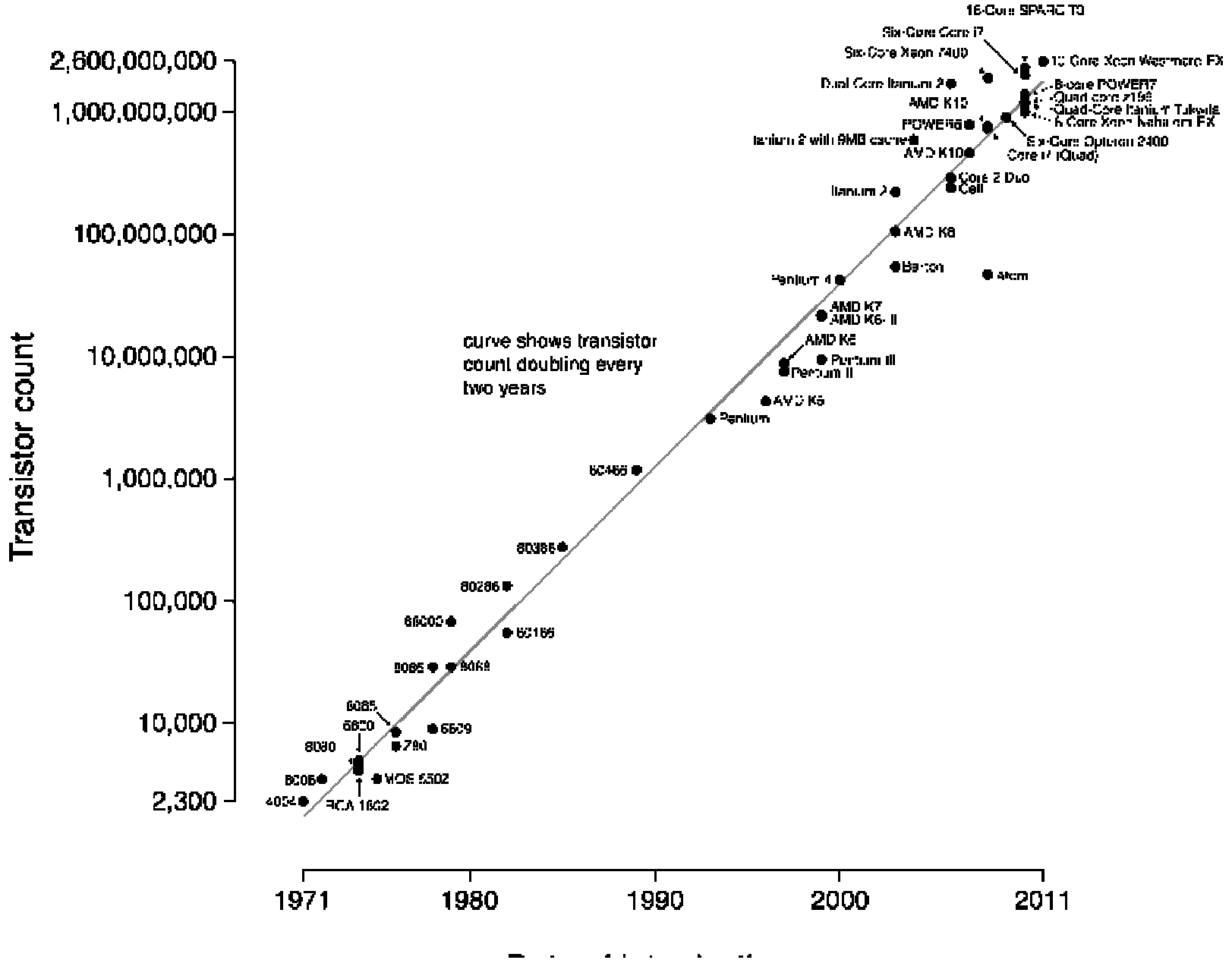
Dans les années 80/90 la conception de processeurs est devenue une science, avec ses lois et ses modèles mathématiques, principalement grâce à des conférences (workshops, symposiums) internationales sur l'architecture des ordinateurs (environ une vingtaine aujourd'hui, généralement bisannuelle)

http://www.wikicfp.com/cfp/call?conference=computer_architecture

CPU Transistor Counts 1971-2008 & Moore's Law



Microprocessor Transistor Counts 1971-2011 & Moore's Law



Notions à illustrer

Les ordinateurs sont devenus TRES compliqués

•Le "pipeline" qui perme de spécialiser certaines parties de la machine dans l'exécution d'une partie (toujours la même) de l'instruction (plus c'est spécialisé, plus c'est rapide). C'est l'équivalent de "travail à la chaîne"

•Les "dépendances" entre instructions

•La segmentation qui permet de protéger matériellement les ressources mémoire d'un programme contre un autre, malintentionné ou simplement maladroit

•La pagination / mémoire virtuelle qui permet, avec une mémoire rapide mais petite et une mémoire grosse mais lente de simuler une mémoire rapide ET grosse (les mémoires c'est comme les administrations : plus c'est gros plus c'est lent).

•L'adressage virtuelle, qui permet à tout programme de s'exécuter en machine comme s'il y était tout seul. Tout les programmes s'exécutent aux mêmes adresses virtuelle, un circuit matériel transforme ces adresses virtuelles en des adresses réelles différentes

Notions à illustrer (cont.)

•L'exécution dans le désordre qui permet d'exécuter simultanément et aussitôt que possible des instructions qui sont écrites pour s'exécuter séquentiellement

•La prédiction (de branchement), qui permet de deviner la décision d'un programme en fonction des décisions passées (le passé est un bon prédicteur de l'avenir).

•L'anticipation, qui consiste à exécuter un bout de programme avant de savoir si c'est vraiment nécessaire. Il faut pouvoir en cas d'erreur annuler totalement l'exécution de ce bout de programme.

•La spéculation, qui consiste à exécuter un bout de programme avant d'en avoir calculé les paramètres (les valeurs probables de certains paramètres sont la valeur précédente ou la valeur précédente plus un).

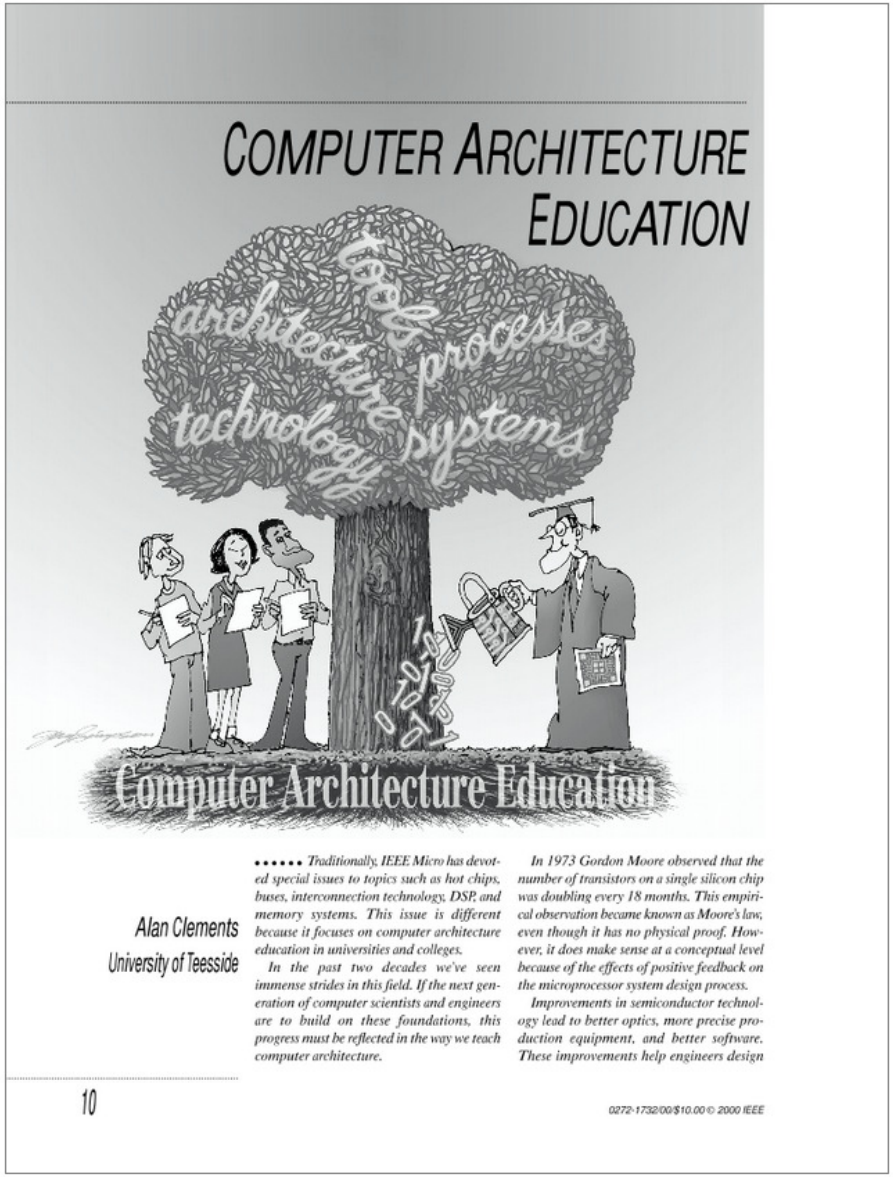
Ces mécanismes sont souvent gourmands en transistors, ce qui donne la loi du "retour sur investissement diminuant" (diminishing return) dite aussi loi des "branches basses".

Computer Architecture educational tools

http://www.ecs.umass.edu/ece/koren/architecture/

- Cache Demonstrator
- Cache-TLB Simulator
- Cache Transient Reloads
- Victim Cache Simulator
- Selective Victim Cache Simulator
- Dual Cache Simulator
- XOR Cache Simulator
- Page Replacement Policies Demo (Javascript)
- New Page Replacement Policies (Java Applet)
- Virtual Memory Simulator
- Memory Interleaving Demo
- Pipelining (w & w/o forwarding)
- Pipelining (Static vs. Dynamic scheduling)
- Scoreboarding
- Tomasulo's algorithm (Java Applet)
- Tomasulo's algorithm (Javascript)
- Reorder Buffer
- Loop Unrolling
- Loop Unrolling in VLIW
- Reservation Table Analyzer
- The SimpleScalar simulator
- The M-Sim (Multi-threaded) simulator
- Branch Prediction
- Branch Target Buffer
- RAID Tutorial
- Vector Processor Simulation

Special issue of IEEE micro



COMPUTER ARCHITECTURE EDUCATION

Computer Architecture Education

Alan Clements
University of Teesside

Traditionally, IEEE Micro has devoted special issues to topics such as hot chips, buses, interconnection technology, DSP, and memory systems. This issue is different because it focuses on computer architecture education in universities and colleges.

In the past two decades we've seen immense strides in this field. If the next generation of computer scientists and engineers are to build on these foundations, this progress must be reflected in the way we teach computer architecture.

In 1973 Gordon Moore observed that the number of transistors on a single silicon chip was doubling every 18 months. This empirical observation became known as Moore's law, even though it has no physical proof. However, it does make sense at a conceptual level because of the effects of positive feedback on the microprocessor system design process.

Improvements in semiconductor technology lead to better optics, more precise production equipment, and better software. These improvements help engineers design

0272-1732/00/$10.00 © 2000 IEEE

Pédagogie et enseignement de l'architecture dans les universités

Conférence et journal (en portugais)

Workshop on Computer Architecture Education

Held in conjunction with

23rd International Conference on
Field-Programmable Logic and Applications

Porto, Portugal
Sunday, 1 September 2013

O ***International Journal of Computer Architecture Education*** **(IJCAE)** é uma publicação mantida pela Comissão Especial de Arquitetura de Computadores e Processamento de Alto Desempenho (CEACPAD) da SBC. Criado em assembléia do SBAC-PAD em 2010, o IJCAE possui como objetivo divulgar trabalhos acadêmicos e científicos da área de Educação em Arquitetura de Computadores. Os artigos submetidos podem ser escritos em Português, Inglês e Espanhol.

Quelques ordinateurs pédagogique

Nous avons choisi 7
"ordinateurs pédagogiques"

Les plus intéressants sont à la fin

- Microprocesseur DLX
- DLX (cont.)
- Pipe-line du DLX (Cont.)

- Simple-CPU
- Simple-CPU (Cont.)
- Simple-CPU (Cont.)
- Simple-CPU (Cont.)

- TiniMips : sous ensemble de MIPS
- TiniMips (cont.)
- TiniMips (cont.)

- Processeur microprogrammé
- Processeur microprogrammé (cont.)

- Micro-processeur LC-3
- Instructions du LC-3 (cont.)

- Little thinker

- Pedagogic Computer
- Pedagogic Computer (cont.)

Microprocesseur DLX

Le DLX (pronounced "Deluxe") est un microprocesseur pédagogique introduit par John L. Hennessy et David A. Patterson en 1981 dans « Computer Architecture: A Quantitative Approach » (5 rééditions). Instructions de 32 bits. (http://fr.wikipedia.org/wiki/DLX).

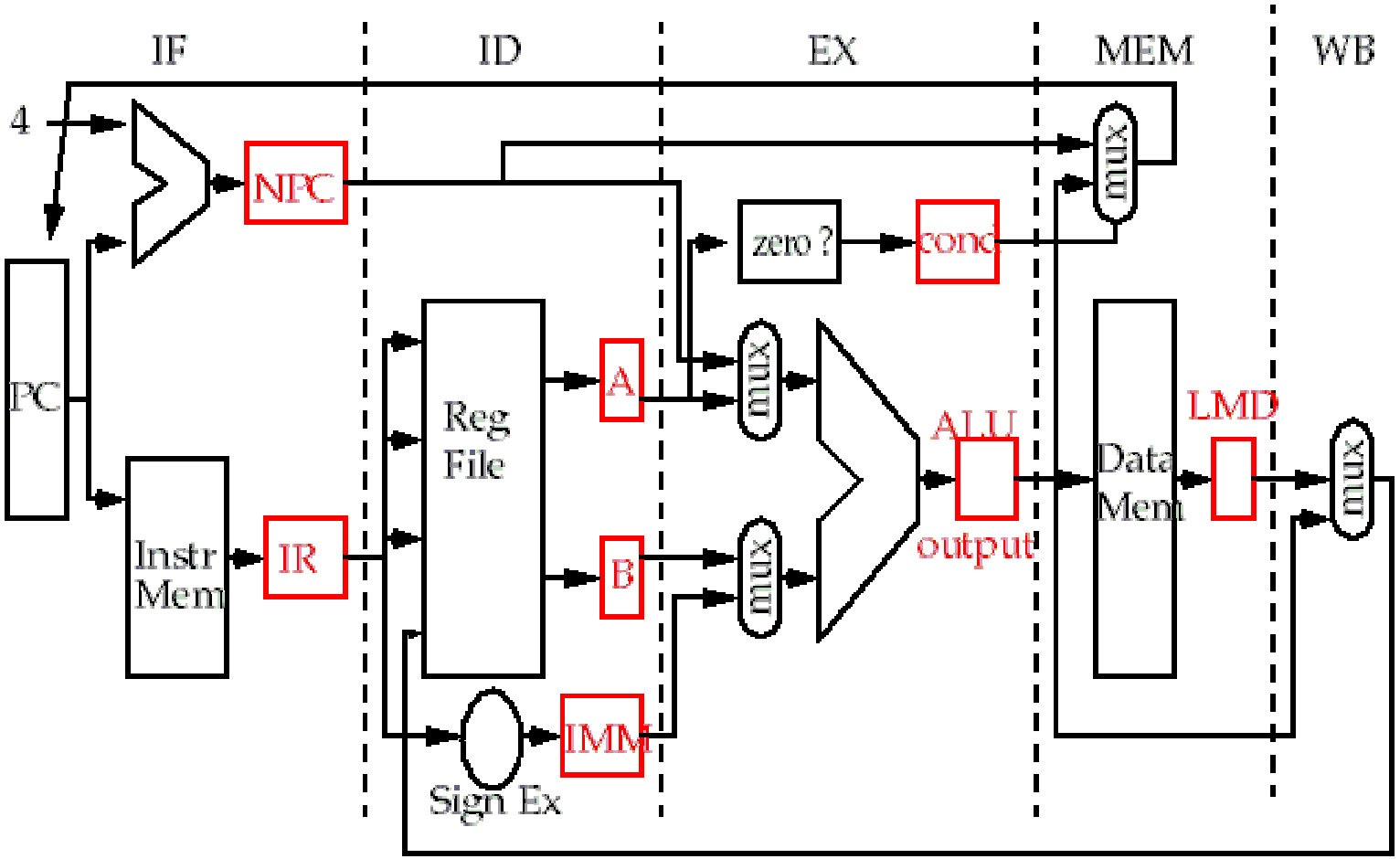
Jeu de 33 instructions

Instr.	Description	Format	Opcode	Operation (C-style coding)
ADD	add	R	0x20	Rd = Rs1 + Rs2
ADDI	add immediate	I	0x08	Rd = Rs1 + extend(immediate)
AND	and	R	0x24	Rd = Rs1 & Rs2
ANDI	and immediate	I	0x0c	Rd = Rs1 & immediate
BEQZ	branch if equal to zero	I	0x04	PC += (Rs1 == 0 ? extend(immediate) : 0)
BNEZ	branch if not equal to zero	I	0x05	PC += (Rs1 != 0 ? extend(immediate) : 0)
J	jump	J	0x02	PC += extend(value)
JAL	jump and link	J	0x03	R31 = PC + 4 ; PC += extend(value)
JALR	jump and link register	I	0x13	R31 = PC + 4 ; PC = Rs1
JR	jump register	I	0x12	PC = Rs1
LHI	load high bits	I	0x0f	Rd = immediate << 16
LW	load woRd	I	0x23	Rd = MEM[Rs1 + extend(immediate)]
OR	or	R	0x25	Rd = Rs1 | Rs2
ORI	or immediate	I	0x0d	Rd = Rs1 | immediate
SEQ	set if equal	R	0x28	Rd = (Rs1 == Rs2 ? 1 : 0)
SEQI	set if equal to immediate	I	0x18	Rd = (Rs1 == extend(immediate) ? 1 : 0)

DLX (cont.)

Instr.	Description	Format	Opcode	Operation (C-style coding)
SLE	set if less than or equal	R	0x2c	Rd = (Rs1 <= Rs2 ? 1 : 0)
SLEI	set if less than or equal to immediate	I	0x1c	Rd = (Rs1 <= extend(immediate) ? 1 : 0)
SLL	shift left logical	R	0x04	Rd = Rs1 << (Rs2 % 8)
SLLI	shift left logical immediate	I	0x14	Rd = Rs1 << (immediate % 8)
SLT	set if less than	R	0x2a	Rd = (Rs1 < Rs2 ? 1 : 0)
SLTI	set if less than immediate	I	0x1a	Rd = (Rs1 < extend(immediate) ? 1 : 0)
SNE	set if not equal	R	0x29	Rd = (Rs1 != Rs2 ? 1 : 0)
SNEI	set if not equal to immediate	I	0x19	Rd = (Rs1 != extend(immediate) ? 1 : 0)
SRA	shift right arithmetic	R	0x07	as SRL & see below
SRAI	shift right arithmetic immediate	I	0x17	as SRLI & see below
SRL	shift right logical	R	0x06	Rd = Rs1 >> (Rs2 % 8)
SRLI	shift right logical immediate	I	0x16	Rd = Rs1 >> (immediate % 8)
SUB	subtract	R	0x22	Rd = Rs1 - Rs2
SUBI	subtract immediate	I	0x0a	Rd = Rs1 - extend(immediate)
SW	store woRd	I	0x2b	MEM[Rs1 + extend(immediate)] = Rd
XOR	exclusive or	R	0x26	Rd = Rs1 ^ Rs2
XORI	exclusive or immediate	I	0x0e	Rd = Rs1 ^ immediate

Pipe-line du DLX (Cont.)



Simple-CPU

http://www.simple-cpu.com/

Le projet *Simple-CPU* a pour objet la création d'un processeur de type RISC libre (diffusé sous licence GPL) caractérisé par :
• Une architecte simple et un jeu d'instruction réduit et classique pour une vocation pédagogique.
• La fourniture d'un ensemble logiciels et système complet pour une utilisation industrielle.
Le processeur est écrit en langage VHDL pour une cible en développement de type FPGA Xilinx. Il est initialement conçu autour d'un kit de démarrage Spartan-3. La chaîne de développement utilisée est entièrement sous Linux.
Simple-CPU est bâti autour d'une architecture 32 bits de type RISC, Load/Store. Il utilise pour communiquer avec les autres périphériques un bus Wishbone lui assurant une interconnectivité maximale avec les IPs libres distribuées par le projet OpenCores.

Simple-CPU est implémenté sur carte Xilinx Spartan, pour 120 €

Simple-CPU (Cont.)

Standard instruction encoding is like this :

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Instruction code								*Destination*					*Operand 1*					*Operand 2*					*Parameters & filler*									

Choix de l'encodage des instructions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction code	Instruction
0	0																															0x0	JMP
0	1																															0x1	CALL
1	1	0																														0x6	JB
1	0	1																														0x5	ADDD
1	0	0	0	0	0	0	0																									0x80	ADDR
1	1	1	1	0	0	0	1																									0xF1	AND
1	1	1	1	0	0	1	0																									0xF2	OR
1	1	1	1	0	0	1	1																									0xF3	XOR
1	1	1	1	0	1	0	0																									0xF4	ROT
1	1	1	1	1	0	0	0																									0xF8	MUL
1	1	1	1	1	0	0	1																									0xF9	DIV
1	1	1	1	1	1	0	0																									0xFC	LOADD
1	1	1	1	1	1	0	1																									0xFD	LOADM
1	1	1	1	1	1	1	0																									0xFE	LOADR
1	1	1	1	1	1	1	1																									0xFF	NOP
1	1	1	0	1	1	0	0																									0xEC	STORE
1	1	1	0	0	0	0	0																									0xE0	CPYB
1	1	1	0	0	1	0	0																									0xE4	PUSH
1	1	1	0	0	1	0	1																									0xE5	POP
1	1	1	0	1	0	0	0																									0xE8	JMPR
1	1	1	0	1	0	0	1																									0xE9	CALLR

Jeu de 21 instructions

Simple-CPU (Cont.)

Incorporation des zones d'opérandes fixes

Les champs d'opérande de destination (Rdd) et d'opérandes sources 1 et 2 (Rnn et Rmm) occupent une place fixe dans les instructions pour permettre un décodage simple des instructions.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INSTRUCTION
CODE INSTRUCTION								DESTINATION					OPÉRANDE 1					OPÉRANDE 2														
0	0																															JMP
0	1																															CALL
1	1	0											RNN																			JB
1	0	1						RDD					RNN																			ADDD
1	0	0	0	0	0	0	0	RDD					RNN					RMM														ADDR
1	1	1	1	0	0	0	1	RDD					RNN					RMM														AND
1	1	1	1	0	0	1	0	RDD					RNN					RMM														OR
1	1	1	1	0	0	1	1	RDD					RNN					RMM														XOR
1	1	1	1	0	1	0	0	RDD					RNN																			ROT
1	1	1	1	1	0	0	0	RDD					RNN					RMM														MUL
1	1	1	1	1	0	0	1	RDD					RNN					RMM														DIV
1	1	1	1	1	1	0	0	RDD																								LOADD
1	1	1	1	1	1	0	1	RDD										RMM														LOADM
1	1	1	1	1	1	1	0	RDD					RNN																			LOADR
1	1	1	1	1	1	1	1																									NOP
1	1	1	0	1	1	0	0						RNN					RMM														STORE
1	1	1	0	0	0	0	0	RDD					RNN																			CPYB
1	1	1	0	0	1	0	0						RNN																			PUSH
1	1	1	0	0	1	0	1	RDD																								POP
1	1	1	0	1	0	0	0						RNN																			JMPR
1	1	1	0	1	0	0	1						RNN																			CALLR

Simple-CPU (Cont.)

Ajout des autres zones

Comme nous l'avons vu, chaque instruction intègre des zones supplémentaires telles que le mode ou des valeurs directes, il nous faut donc maintenant les placer parmi l'espace non encore occupé.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INSTRUCTION
CODE INSTRUCTION								DESTINATION					OPÉRANDE 1					OPÉRANDE 2														
0	0	ADRESSE ABSOLUE																														JMP
0	1	ADRESSE ABSOLUE																														CALL
1	1	0	MODE				V	BIT					RNN					ADRESSE RELATIVE														JB
1	0	1				VALUE		RDD					RNN					VALUE														ADDD
1	0	0	0	0	0	0	0	RDD					RNN					RMM			MODE											ADDR
1	1	1	1	0	0	0	1	RDD					RNN					RMM														AND
1	1	1	1	0	0	1	0	RDD					RNN					RMM														OR
1	1	1	1	0	0	1	1	RDD					RNN					RMM														XOR
1	1	1	1	0	1	0	0	RDD					RNN					$TEP			DIR	MODE										ROT
1	1	1	1	1	0	0	0	RDD					RNN					RMM			MODE											MUL
1	1	1	1	1	0	0	1	RDD					RNN					RMM			MODE	TYPE										DIV
1	1	1	1	1	1	0	0	RDD					VALEUR DIRECTE																MODE			LOADD
1	1	1	1	1	1	0	1	RDD					RNN								MODE											LOADM
1	1	1	1	1	1	1	0	RDD					RNN								MODE											LOADR
1	1	1	1	1	1	1	1																									NOP
1	1	1	0	1	1	0	0						RNN					RMM			MODE											STORE
1	1	1	0	0	0	0	0	RDD					RNN					DEST BIT			MODE						$RC BIT					CPYB
1	1	1	0	0	1	0	0						RNN																			PUSH
1	1	1	0	0	1	0	1	RDD																								POP
1	1	1	0	1	0	0	0						RNN																			JMPR
1	1	1	0	1	0	0	1						RNN																			CALLR

Les zones non affectées ne sont pas utilisées, elles seront par défaut mise à zéro. Elles pourront aussi être utilisées par la suite pour optimiser le décodage des signaux de l'ALU en complément du mode. Nous verrons celà lors de la phase d'implémentation. L'association entre les modes et leur valeur n'est pas décrite ici mais vous la trouverez dans le document décrivant le jeu d'instruction.

TiniMips : sous ensemble de MIPS

David Harris at Harvey Mudd College, Claremont, California

http://www.eng.utah.edu/~cs6710/slides/mipsx2.pdf
http://www3.hmc.edu/~harris/cmosvlsi/4e/index.html

MIPS Architecture
* Example: subset of MIPS processor architecture, Drawn from Patterson & Hennessy
MIPS is a 32-bit architecture with 32 registers
• Consider 8-bit subset using 8-bit datapath
• Only implement 8 registers ($0 - $7)
• $0 hardwired to 00000000
• 8-bit program counter

But : faire concevoir, simuler, vérifier, TiniMips en VHDL

TiniMips (cont.)

Jeu de 10 instructions

Table 1.7 MIPS instruction set (subset supported)

Instruction	Function		Encoding	op	funct
`add $1, $2, $3`	addition:	$1 → $2 + $3	R	000000	100000
`sub $1, $2, $3`	subtraction:	$1 → $2 – $3	R	000000	100010
`and $1, $2, $3`	bitwise and:	$1 → $2 and $3	R	000000	100100
`or $1, $2, $3`	bitwise or:	$1 → $2 or $3	R	000000	100101
`slt $1, $2, $3`	set less than:	$1 → 1 if $2 < $3 $1 → 0 otherwise	R	000000	101010
`addi $1, $2,`	add immediate:	$1→ $2 + imm	I	001000	n/a
`beq $1, $2, imm`	branch if equal:	PC → PC + imm[a]	I	000100	n/a
`j destination`	jump:	PC_destination[a]	J	000010	n/a
`lb $1, imm($2)`	load byte:	$1 → mem[$2 + imm]	I	100000	n/a
`sb $1, imm($2)`	store byte:	mem[$2 + imm] → $1	I	110000	n/a

TiniMips (cont.)

Calcul du n-ième nombre de la suite de Fibbonaci (sic.)

```
int fib(void)
{
      int n=8;
      int f1 = 1;
      int f2 = -1;

      while (n != 0)
      {
            f1=f1+f2;
            f2=f1-f2;
            n=n-1;
      }
return f1;
}
```

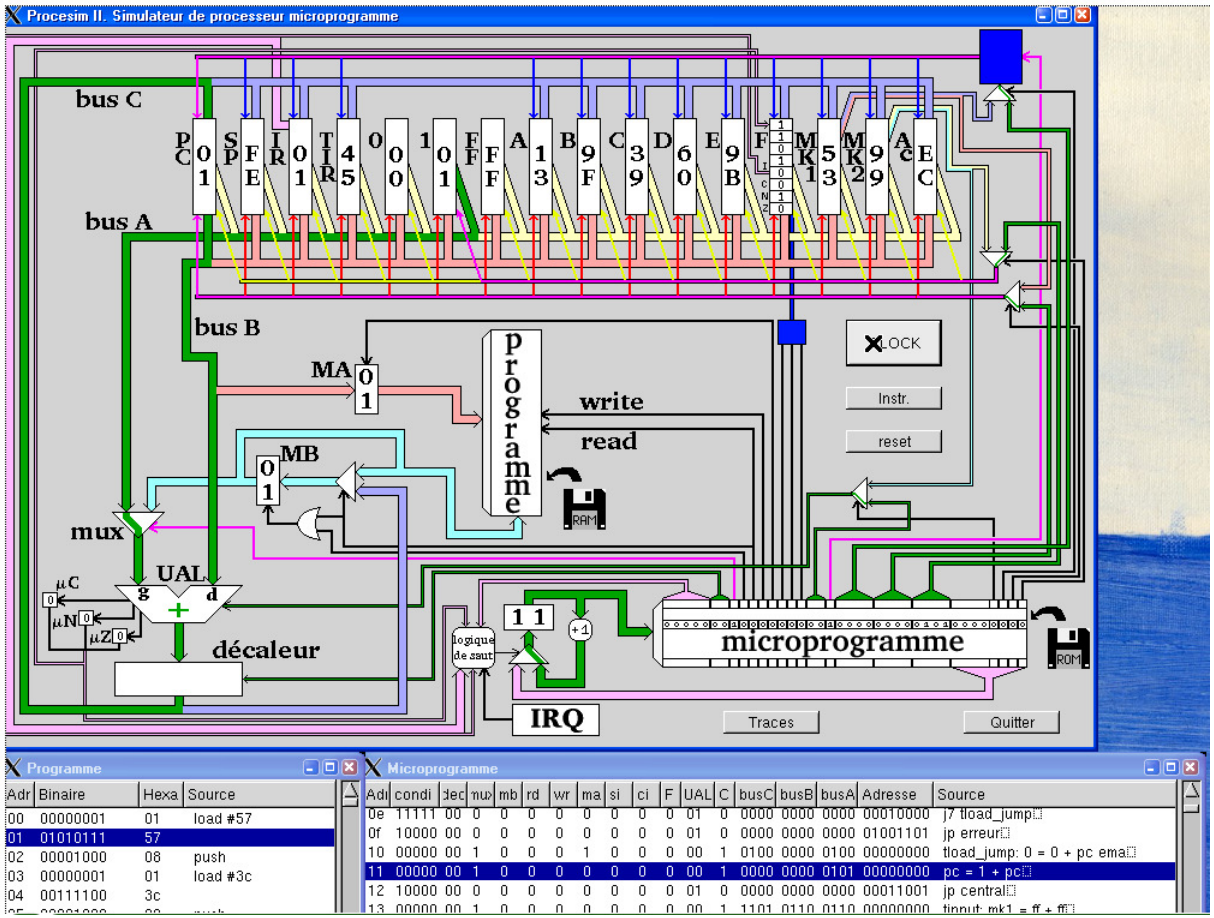
```
addi $3, $0, 8
addi $4, $0, 1
addi $5, $0, -1
beq $3, $0, end
add $4, $4, $5
sub $5, $4, $5
addi $3, $3, -1
j loop
sb $4, 255($0)
or $3, $4, $0
j instr2
```

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent
« Un homme met un couple de lapins dans un lieu isolé de tous les côtés par un mur. Combien de couples obtient-on en un an si chaque couple engendre tous les mois un nouveau couple à compter du troisième mois de son existence ? » (Wikipedia)

Processeur microprogrammé

Denis Bouhineau, Maître de conférences (UJF), enseignant à Polytech

http://www.youtube.com/watch?v=bVSkvr89zoo
http://www.noe-kaleidoscope.org/people/DenisB/Enseignement/Architecture/
http://www.noe-kaleidoscope.org/public/people/DenisB/Enseignement/Architecture/DemoProcesim.htm



Processeur microprogrammé (cont.)

Il n'y a pas de jeu d'instruction !

… En cette journée de grève, privé d'étudiant, j'ai construit une petite démonstration avec le logiciel ProceSim (et wink pour le coté démo) que nous utilisons pour visualiser le passage entre le niveau circuit et niveau langage machine. (Denis Bouhineau)

ProceSim est également utilisé par Pascal Sicard (thèse avec Pr. Saucier)

Sur demande, je peux faire un petit rappel sur la microprogrammation

Micro-processeur LC-3

Olivier Carton, Laboratoire d'Informatique Algorithmique, Paris 7

http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/Cours/LC3/

Le microprocesseur LC-3 est à vocation pédagogique. Il n'existe pas de réalisation concrète de ce processeur mais il existe des simulateurs permettant d'exécuter des programmes.
L'intérêt de ce microprocesseur est qu'il constitue un bon compromis de complexité. Il est suffisamment simple pour qu'il puisse être appréhendé dans son ensemble et que son schéma en portes logiques soit accessible. Il comprend cependant les principaux mécanismes des microprocesseurs (appels système, interruptions) et son jeu d'instructions est assez riche pour écrire des programmes intéressants (O. Carton)

Instructions du LC-3 (cont.)

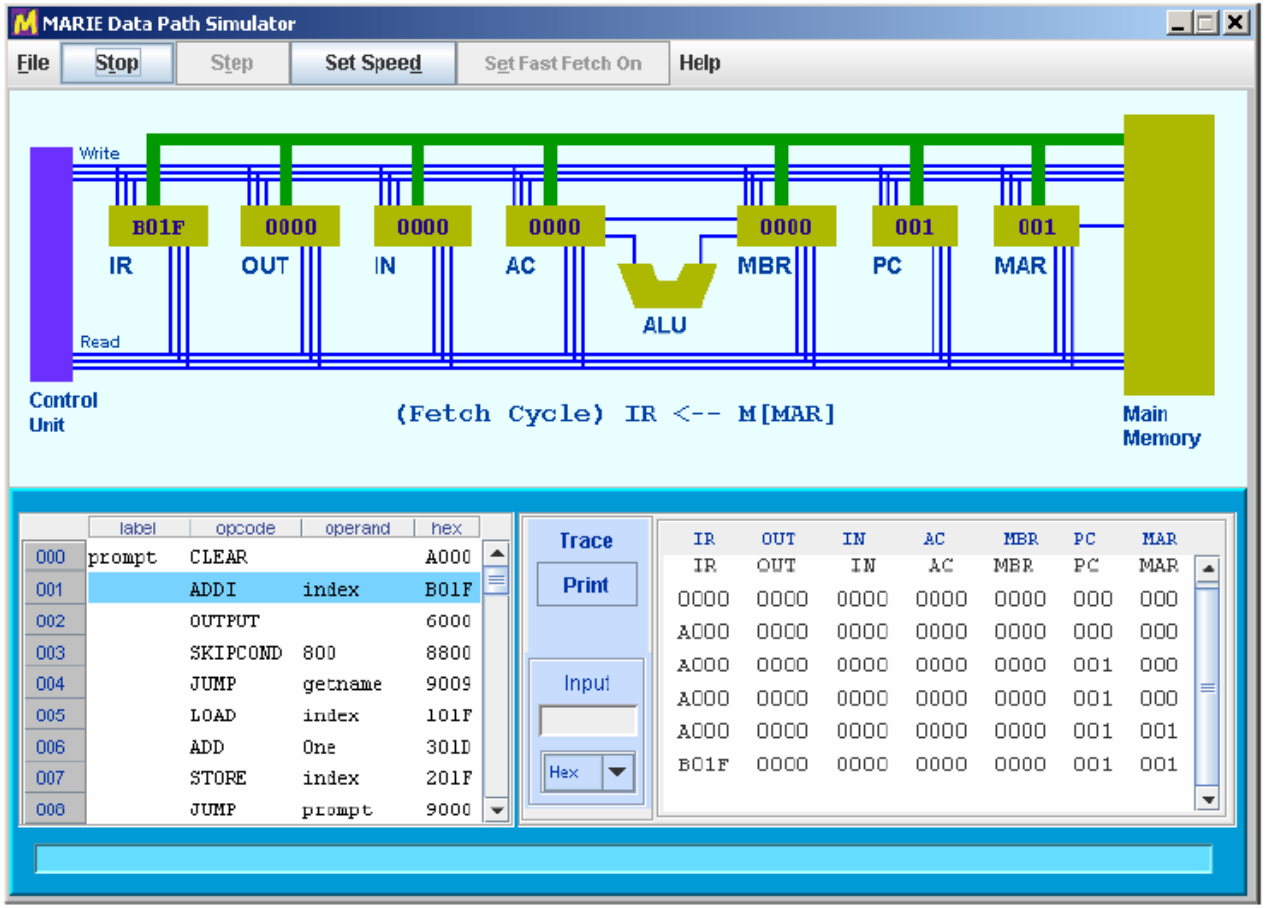
Jeu de 20 instructions

Syntaxe	Action	nzp	Codage															
			Op-code				Arguments											
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT DR,SR	DR ← not SR	*	1	0	0	1	DR			SR			1	1	1	1	1	1
ADD DR,SR1,SR2	DR ← SR1 + SR2	*	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*	0	0	0	1	DR			SR1			1	Imm5				
AND DR,SR1,SR2	DR ← SR1 and SR2	*	0	1	0	1	DR			SR1			0	0	0	SR2		
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*	0	1	0	1	DR			SR1			1	Imm5				
LEA DR,label	DR ← PC + SEXT(PCoffset9)	*	1	1	1	0	DR			PCoffset9								
LD DR,label	DR ← mem[PC + SEXT(PCoffset9)]	*	0	0	1	0	DR			PCoffset9								
ST SR,label	mem[PC + SEXT(PCoffset9)] ← SR		0	0	1	1	SR			PCoffset9								
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*	0	1	1	0	DR			BaseR			Offset6					
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR		0	1	1	1	SR			BaseR			Offset6					
LDI DR,label	DR ← mem[mem[PC + SEXT(PCoffset9)]]	*	1	0	1	0	DR			PCoffset9								
STI SR,label	mem[mem[PC + SEXT(PCoffset9)]] ← SR		1	0	1	1	SR			PCoffset9								
BR[n][z][p] label	Si (cond) PC ← PC + SEXT(PCoffset9)		0	0	0	0	n	z	p	PCoffset9								
NOP	No Operation		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JMP BaseR	PC ← BaseR		1	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RET (≡ JMP R7)	PC ← R7		1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
JSR label	R7 ← PC; PC ← PC + SEXT(PCoffset11)		0	1	0	0	1	PCoffset11										
JSRR BaseR	R7 ← PC; PC ← BaseR		0	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RTI	cf. interruptions		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TRAP Trapvect8	R7 ← PC; PC ← mem[Trapvect8]		1	1	1	1	0	0	0	0	Trapvect8							
Réservé			1	1	0	1												

MARIE (Machine Architecture that is Really Intuitive and Easy)

Linda Null and Julia Lobur, Pennsylvania State University

http://www.amazon.com/The-Essentials-Computer-Organization-Architecture/dp/1449600069



Marie (Cont.)

Jeu de 15 instructions

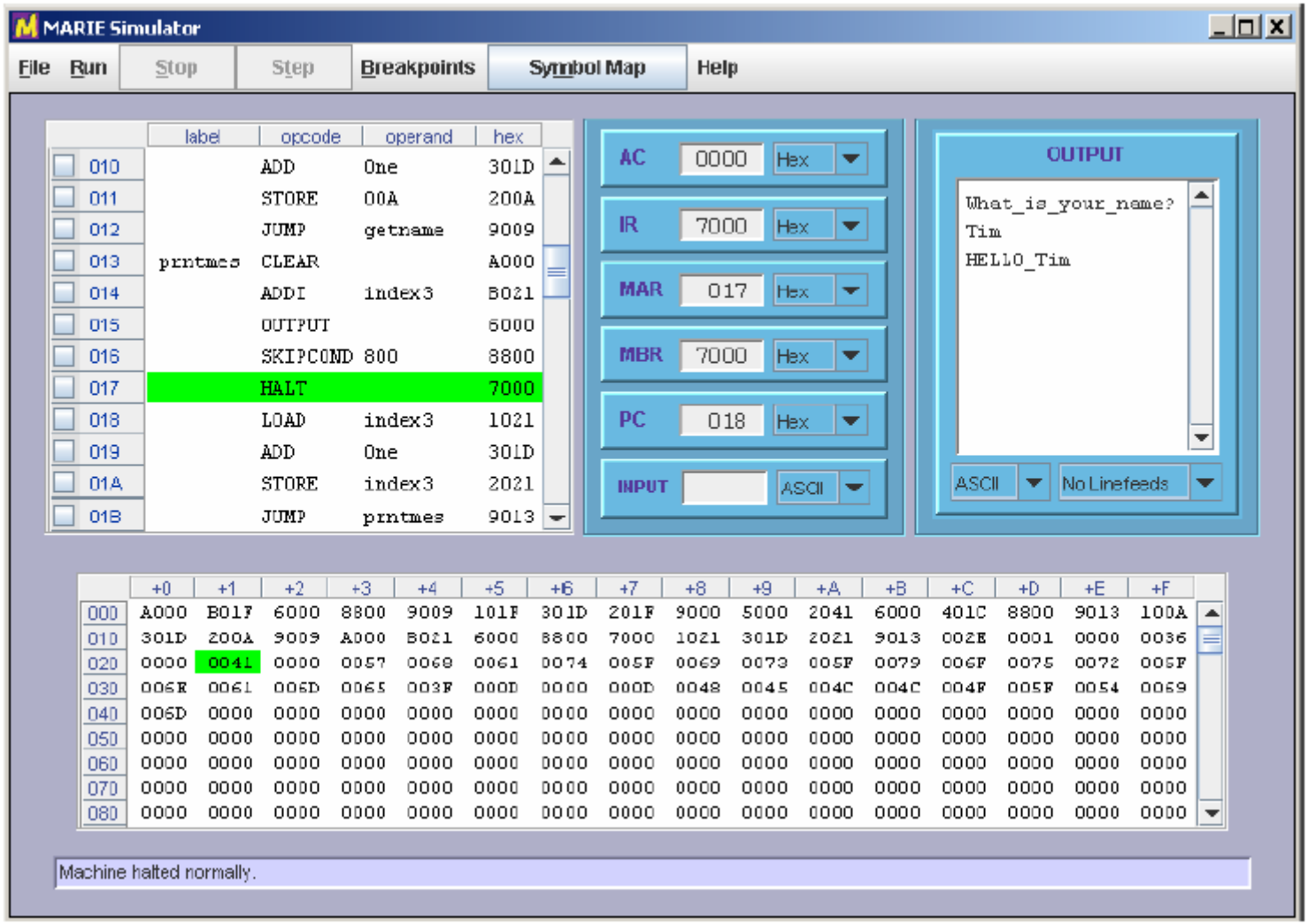
Opcode	Instruction	RTN
0000	JnS *X*	MBR ⟵ PC MAR ⟵ *X* M[MAR] ⟵ MBR MBR ⟵ *X* AC ⟵ 1 AC ⟵ AC + MBR PC ⟵ AC
0001	Load *X*	MAR ⟵ *X* MBR ⟵ M[MAR], AC ⟵ MBR
0010	Store *X*	MAR ⟵ *X*, MBR ⟵ AC M[MAR] ⟵ MBR
0011	Add *X*	MAR ⟵ *X* MBR ⟵ M[MAR] AC ⟵ AC + MBR
0100	Subt *X*	MAR ⟵ *X* MBR ⟵ M[MAR] AC ⟵ AC - MBR
0101	Input	AC ⟵ InREG
0110	Output	OutREG ⟵ AC
0111	Halt	
1000	Skipcond	If IR[11–10] = 00 then If AC < 0 then PC ⟵ PC + 1 Else If IR[11–10] = 01 then If AC = 0 then PC ⟵ PC + 1 Else If IR[11–10] = 10 then If AC > 0 then PC ⟵ PC + 1
1001	Jump *X*	PC ⟵ IR[11–0]
1010	Clear	AC 0
1011	AddI *X*	MAR ⟵ X MBR ⟵ M[MAR] MAR ⟵ MBR MBR ⟵ M[MAR] AC ⟵ AC + MBR
1100	JumpI *X*	MAR ⟵ *X* MBR ⟵ M[MAR] PC ⟵ MBR

MARIE exhibits the classical von Neumann design, and includes a program counter, an accumulator, an instruction register, 4096 bytes of memory, and two addressing modes. Assembly language programming is introduced to reinforce the concepts of instruction format, instruction mode, data format, and control.

MarieSim écrit en Java. Source disponible.

Marie (Cont.)

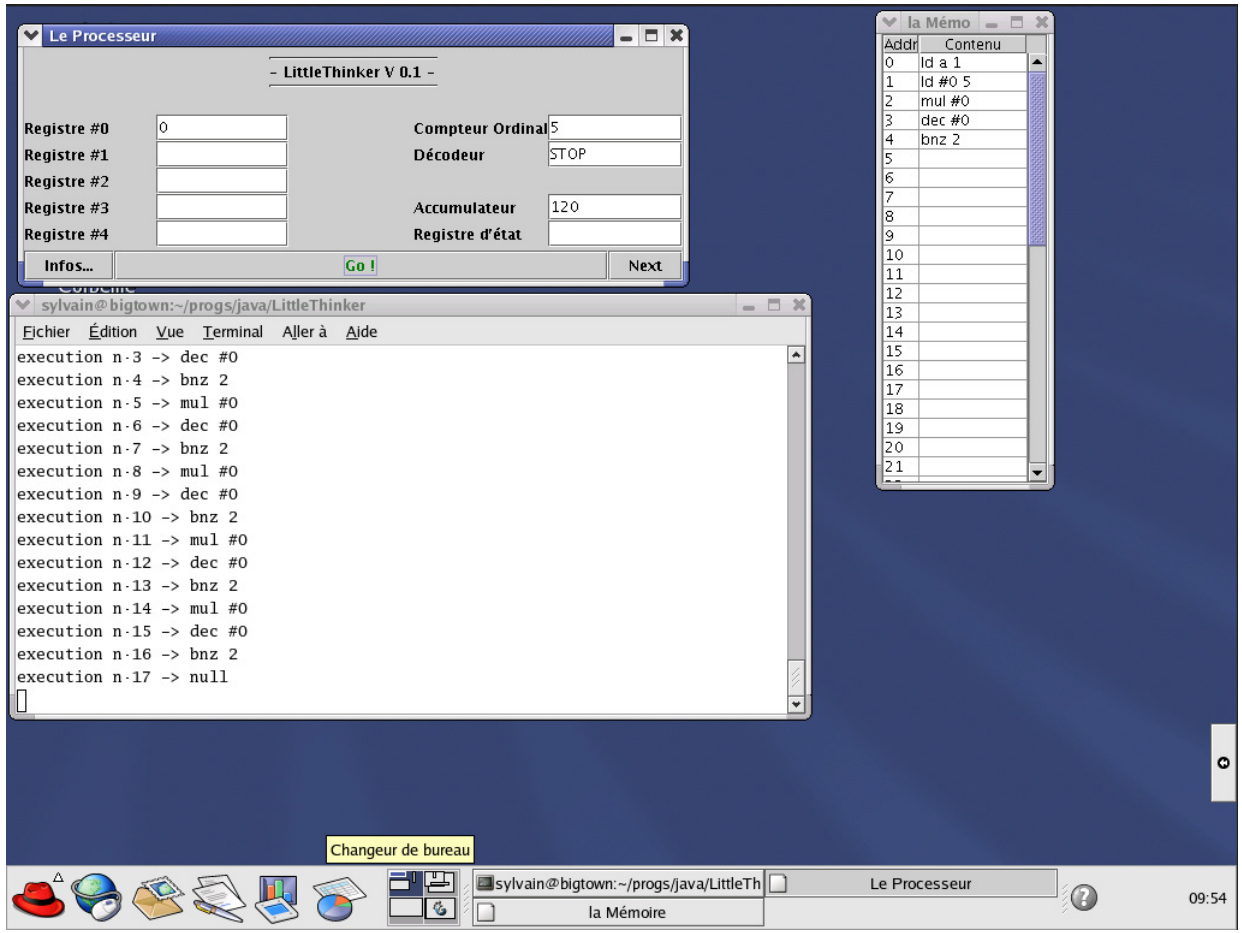
MarieSim, has a user-friendly GUI that allows students to: (1) create and edit source code; (2) assemble source code into machine object code; (3) run machine code; and, (4) debug programs.



Little thinker

Sylvain Cherrier, enseignant à l'Université de Marne-la-Vallée

http://sylvain.cherrier.free.fr/LT/LittleThinker0.1.pdf
http://sylvain.cherrier.free.fr/LT.html

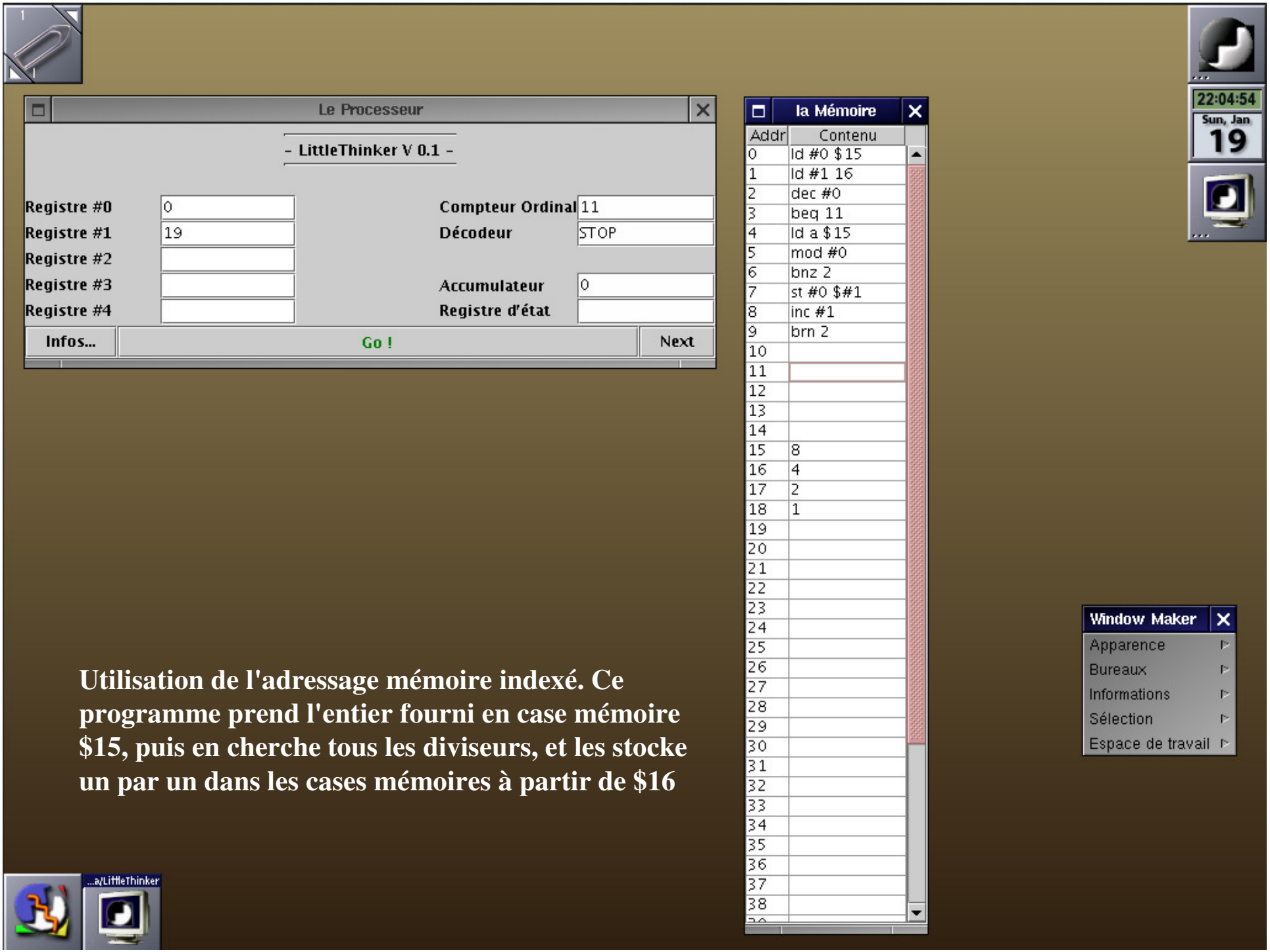


Réalisé en Java
(sources disponibles)

3 fenêtres:
1- État processeur
2- Mémoire (instructions)
3- Trace d'exécution

Jeu de 19 instructions

PAS DE CODAGE !



Utilisation de l'adressage mémoire indexé. Ce programme prend l'entier fourni en case mémoire $15, puis en cherche tous les diviseurs, et les stocke un par un dans les cases mémoires à partir de $16

Experiences in the design of a Pedagogic Computer

E. Pastor, F. Sanchez, A. Del Corral
Universitat Politècnica de Catalunya. Barcelona

http://biblioteca.universia.net/html_bura/ficha/params/title/rudimentary-machine-experiences-in-the-design-of-pedagogic-computer/id/49609881.html
ftp://ftp.ac.upc.es/pub/archives/mr (ne marche pas !!!)
http://www.ncsu.edu/wcae/ISCA1998/pastor.pdf

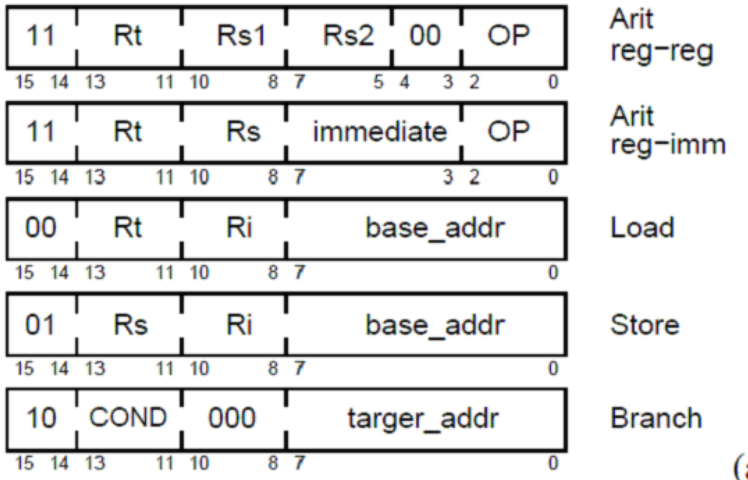
Jeu d'instructions (15 instructions)

Instruction	Operation
ADDI Rs, #inmediate, Rt	Rt := Rs + inmediate
SUBI Rs, #inmediate, Rt	Rt := Rs − inmediate
ADD Rs1, Rs2, Rt	Rt := Rs1 + Rs2
SUB Rs1, Rs2, Rt	Rt := Rs1 − Rs2
ASR Rs, Rt	Rt := Rs >> 1
AND Rs1, Rs2, Rt	Rt := Rs1 ∧ Rs2
LOAD base_addr(Ri), Rt	Rt := M[base_addr + Ri]
STORE Rs, base_addr(Ri)	M[base_addr + Ri] := Rs

Instruction	Branch condition	
BR target_addr	unconditional	1
BEQ target_addr	equal	Z
BL target_addr	less	N
BLE target_addr	less or equal	N ∨ Z
BNE target_addr	different	$\overline{Z}$
BGE target_addr	greater or equal	$\overline{N}$
BG target_addr	greater	$\overline{N \vee Z}$

Pedagogic Computer (cont.)

Jeu d'instructions et format (cont.)



(a)

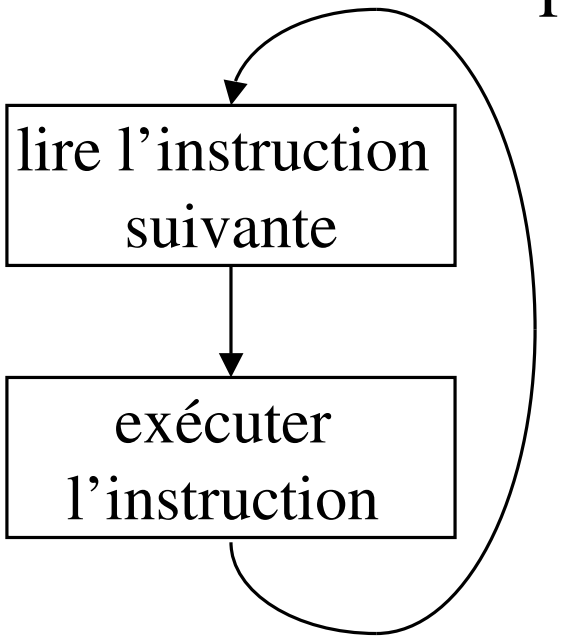
Arithmetic instruction	OP
Add reg-immediate	000
Sub reg-immediate	001
Add reg-reg	100
Sub reg-reg	101
Shift right	110
Logic AND	111

(b)

Branch instructions	COND
Unconditional	000
Equal	001
Less	010
Less or equal	011
Different	101
Greater or equal	110
Greater	111

(c)

Conclusion



Pas d' "Ordinateur Pédagogique" pour lycéen

- Maquette ET logiciel ?
- Jeu d'instruction extensible ?
- Instructions codées (en binaire) ?
- Sous-ensemble d'un VRAI processeur ?
- Instructions arithmétiques complexes (mult, div, mod) ?
- Mode adressage (indirect, indexé, pile, immédiat, …)
- Outils de mise au point (trace, assembleur dynamique, ..)