

TỔNG QUAN	DỊCH LUẬN LÝ	QUÁ TRÌNH PHÁT TRIỂN	5 THÀNH PHẦN CƠ BẢN:																												
1_ Tổng quan máy tính 2_Biểu diễn số nguyên 3_Biểu diễn số thực 4,5,6_Lập trình Hợp Ngữ 7_Mạch Logic 8_Thiết kế CPU 9_Pipeline 10_Bộ nhớ 11_Hệ thống nhập xuất	-Dịch trái (sl) – shift left logical: Thêm vào các bit 0 bên phải -Dịch phải (srl – shift right logical): Thêm vào các bit 0 bên trái DỊCH SỐ HỌC <small>0 co shr shr</small> -Dịch phải (sra – shift right arithmetic): Thêm các bit = giá trị bit đầu bên trái	Thế hệ Khoảng thời gian Công nghệ 1.1940 – 1956 Vacuum tubes (đèn chân không) 2. 1956 – 1963 Transistors (linh kiện bán dẫn) 3. 1964 – 1971 Integrated Circuits (vi mạch tích hợp) 4. 1971 – nay Microprocessors (vi xử lý) 5. Tương lai Parallel Processing ĐỊNH LUẬT MOORE Số lượng transistor tích hợp trong 1 IC tăng gấp đôi mỗi 1.5 năm(18 tháng).	Input, output, memory, processor, datapath, control. <Bộ xử lý, bộ nhớ chính, hệ thống kết nối , thiết bị nhập/ xuất> WAFER (ĐẾ CHIP): Tấm silicon mỏng đã được cấy vật liệu khác nhau để tạo ra những vi mạch -Có kích thước trung bình từ 25,4mm (1inch) – 200mm (7.9inch). CHIP: Có thể hiểu là mạch tích hợp (Integrated Circuit) gắn trên đế chip (wafer) nhằm xử lý các công việc trên máy tính. -Chip có kích thước rất nhỏ nhưng có thể chứa hàng chục triệu transistor, số lượng transistor càng lớn thì tốc độ truyền và xử lý tin hiệu càng nhanh -Hiện nay có các loại chip xử lý: 4, 8, 16, 32, 64 bit																												
-Nhiệm vụ cơ bản nhất của CPU là phải thực hiện các lệnh được yêu cầu, gọi là instruction -Các CPU sẽ sử dụng các tập lệnh (instruction set) khác nhau để có thể giao tiếp với nó KÍCH THỦC LỆNH BỊ ÁNH HƯỚNG BỚI: +Cấu trúc đường truyền bus +Kích thước và tổ chức bộ nhớ +Tốc độ CPU -Giải pháp tối ưu lệnh: +Dùng lệnh có kích thước ngắn, mỗi lệnh chỉ nên được thực thi trong đúng 1 chu kỳ CPU +Dùng bộ nhớ cache BỘ LỆNH MIPS Được xây dựng theo kiến trúc (RISC) với 4 nguyên tắc: +Càng đơn giản, càng ổn định. +Càng nhỏ gọn, xử lý càng nhanh. +Tăng tốc xử lý cho những trường hợp thường xuyên xảy ra +Thiết kế đòi hỏi sự thỏa hiệp tốt	- x SHL y = x . 2 ⁱ ; x SHR y = x / 2 ⁱ - AND dùng để tất bit (AND với 0 luôn = 0); OR dùng để bất bit (OR với 1 luôn = 1) - XOR, NOT dùng để đảo bit (XOR với 1 = đảo bit đó) - x AND 0 = 0; x XOR x = 0 - Lấy giá trị tại bit thứ i của x: (x SHR i) AND 1; - Gán giá trị 1 tại bit thứ i của x: (1 SHL i) OR x; - Gán giá trị 0 tại bit thứ i của x: NOT(1 SHL i) AND x; Đảo bit thứ i của x: (1 SHL i) XOR x	BIỂU DIỄN SỐ THỰC SAU THEO DẠNG SỐ CHẤM ĐỘNG CHÍNH XÁC ĐƠN (32 bit): X = -5.25 Bước 1: Đổi X sang hệ nhị phân X = -5.25 ₁₀ = -101.01 ₂ kép:64 (1-11-52) Bước 2: Chuẩn hóa theo dạng $\pm 1.F * 2^E$ X = -5.25 = -101.01 = -1.0101 * 2 ² Bước 3: Biểu diễn Floating Point - Số âm: bit đầu Sign = 1 - Số mũ E = 2 -> Phần mũ exponent với số thừa K=127 được biểu diễn:->Exponent = E + 127 = 2 + 127 = 129 ₁₀ = 1000 0001 ₂ - Phần định trị = 0101 0000 0000 0000 0000 0000 (Thêm 1 số 0 cho dù 23 bit) - Kết quả nhận được: 1 1000 0001 0101 0000 0000 0000 0000 0000 1-8-23	CHIPSET: là tập hợp nhiều chip gắn kết lại với nhau trên cùng 1 đế chip (wafer) để xử lý nhiều công việc trên máy tính -Một số chipset thông dụng: CPU: Đơn vị xử lý trung tâm; GPU: Đơn vị xử lý đồ họa trên máy; RAM: Bộ nhớ truy cập tức thời chuyên phục vụ cho CPU; BẢN CÀU BẮC (tích hợp trên mainboard): Hỗ trợ truyền thông tin cho CPU, RAM, nắp sát CPU (Hệ thống Mainboard AMD không có chipset này vì được tích hợp ngay trên CPU); BẢN CÀU NAM (tích hợp trên mainboard): Quản lý thiết bị ngoại vi như HDD, Mouse, Keyboard,...Nắp cuối mainboard																												
-A=B beq \$s0, \$s1, label #if (\$s0==\$s1) goto label -A!=B bne \$s0, \$s1, label #if (\$s0!=\$s1) goto label -A < B slt \$t0, \$s0, \$s1 # if (a < b) then \$t0 = 1 bne \$t0, \$0, Label # if (a < b) then goto Label -A > B slt \$t0, \$s1, \$s0 # if (b < a) then \$t0 = 1 bne \$t0, \$0, Label # if (b < a) then goto Label -A ≥ B slt \$t0, \$s0, \$s1 # if (a < b) then \$t0 = 1 beq \$t0, \$0, Label # if (a ≥ b) then goto Label -A ≤ B slt \$t0, \$s1, \$s0 # if (b < a) then \$t0 = 1 beq \$t0, \$0, Label # if (b ≥ a) then goto Label Re co dk: j label (= beq \$0,\$0,label)	NGÔN NGỮ MÁY (MACHINE LANGUAGE) -Ngôn ngữ máy cho phép người lập trình đưa ra các hướng dẫn đơn giản mà bộ vi xử lý (CPU) có thể thực hiện được ngay:+Các hướng dẫn này được gọi là chỉ thị / lệnh (instruction) hoặc mã máy (machine code) +Mỗi bộ vi xử lý (CPU) có 1 ngôn ngữ riêng, gọi là bộ lệnh (instruction set) -Trong cùng 1 dòng vi xử lý (processor family) bộ lệnh gần giống nhau INSTRUCTION -Là dãy bit chứa yêu cầu mà bộ vi xử lý trong CPU (ALU) phải thực hiện -Instruction gồm 2 thành phần: +Mã lệnh (opcode): thao tác cần thực hiện +Thông tin về toàn hàng (operand): các đối tượng bị tác động bởi thao tác chứa trong mã lệnh	- Bus (hệ thống dẫn đường): liên kết các thành phần lại với nhau. Gồm có: bus cục bộ (đường dẫn cục bộ nối các thành phần bên trong 1 thiết bị) và bus hệ thống (hệ thống dẫn đường liên quan các thiết bị quan trọng như CPU, bộ nhớ và các mạch ra vào). - PHÂN LOẠI: bus địa chỉ, bus dữ liệu, bus điều khiển. - NGẮT: là dừng chương trình đang thực hiện để thực hiện 1 chương trình khác. Phân loại : 3 loại + Ngắt cứng : sinh ra do các tín hiệu INTR (ngắt che được) hay NMI (ko che được), gồm có 2 loại: ngắt che được và ngắt không che được.+ Ngắt mềm : sinh ra do câu lệnh INT. + Ngắt tự động (ngoại lệ) : sinh ra do thực hiện các lệnh của CPU như chia 0, đặt cờ ngắt....																													
NGUYÊN TẮC LƯU ĐỮA LIỆU TRONG BỘ NHỚ -MIPS thao tác và lưu trữ dữ liệu trong bộ nhớ theo 2 nguyên tắc: +ALIGNMENT RESTRICTION: Các đối tượng lưu trong bộ nhớ (tùy nhớ) phải bắt đầu tại địa chỉ là bội số của kích thước đối tượng. -Mỗi từ nhớ có kích thước là 32 bit = 4 byte = kích thước lưu trữ của 1 thanh ghi trong CPU -Như vậy, từ nhớ phải bắt đầu tại địa chỉ là bội số của 4 +BIG ENDIAN: (MIPS, X86:thuan: 0-12,1-34,2-56 LittleEndian: 0-78,1-56,2-34,3-12)	ISA (Instruction Set Architecture) -Tập lệnh dành cho những bộ vi xử lý có kiến trúc tương tự nhau -MỘT SÓ ISA THÔNG DỤNG: -Đòng vi xử lý 80x86 (gọi tắt x86) của Intel +IA-16: Đòng xử lý 16 bit +IA-32: Đòng xử lý 32 bit +IA-64: Đòng xử lý 64 bit -MIPS: Dùng rất nhiều trong hệ thống nhúng (embedded system) -PowerPC của IBM -THIẾT KẾ ISA: CISC & RISC +Có 2 trường phái thiết kế bộ lệnh: +Complete Instruction Set Computer (CISC): bộ lệnh gồm rất nhiều lệnh, từ đơn giản đến phức tạp -Reduced Instruction Set Computer (RISC): bộ lệnh chỉ gồm các lệnh đơn giản	S Exp Significand (Fraction) -Largest positive normalized number: +1.[23 số 1] * 2 ¹²⁷ 0 1111 1110 1111 1111 1111 1111 111 -Smallest positive normalized number: +1.[23 số 0] * 2 ⁻¹²⁶ 0 0000 0001 0000 0000 0000 0000 0000 000 -Tương tự cho số negative (số âm) -Largest positive denormalized number: +0.[23 số 1] * 2 ⁻¹²⁷ 0 0000 0000 1111 1111 1111 1111 111 Tuy nhiên IEEE 754 quy định là +0.[23 số 1] * 2 ⁻¹²⁶ vì muốn tiền gần hơn với "Smallest positive normalized number = +1.[23 số 0] * 2 ⁻¹²⁶ , -Smallest positive denormalized number: +1.[22 số 0] * 2 ⁻¹²⁷ 0 0000 0000 0000 0000 0000 0000 0000 0001. Tuy nhiên IEEE 754 quy định là +0.[22 số 0] * 2 ⁻¹²⁶ -Tương tự cho số negative (số âm)																													
CÓ 3 FORMAT LỆNH TRONG MIPS: -R-format: Dùng trong các lệnh tính toán số học (add, sub, and, or, nor, sll, srl, sra...) <table border="1"><tr><td>6 bits</td><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr><tr><td>opcode</td><td>hex</td><td>rs</td><td>dec</td><td>rt</td><td>rd</td><td>shmat funct</td></tr></table> -I-format: Dùng trong các lệnh thao tác với hàng số, chuyển dữ liệu với bộ nhớ, rõ nhánh sw:2b, lw:23 <table border="1"><tr><td>6 bits</td><td>5</td><td>5</td><td>16</td><td>immediate</td></tr><tr><td>opcode</td><td>rs</td><td>rt</td><td></td><td></td></tr></table> -J-format: Dùng trong các lệnh nhảy (jump – C: goto) <table border="1"><tr><td>6 bits</td><td>26</td><td>target address</td></tr><tr><td>opcode</td><td></td><td></td></tr></table>	6 bits	5	5	5	5	5	6	opcode	hex	rs	dec	rt	rd	shmat funct	6 bits	5	5	16	immediate	opcode	rs	rt			6 bits	26	target address	opcode			COMPLIER -Trình biên dịch ngôn ngữ cấp cao -> hợp ngữ -Compiler phần thuộc vào: +Ngôn ngữ cấp cao được biên dịch +Kiến trúc hệ thống phân cứng bên dưới mà nó đang chạy ASSEMBLER -Trình biên dịch hợp ngữ -> ngôn ngữ máy -Một bộ vi xử lý (đi kèm 1 bộ lệnh xác định) có thể có nhiều Assembler của nhiều nhà cung cấp khác nhau chạy trên các OS khác nhau -Assembly program phụ thuộc vào Assembler mà nó sử dụng (do các mờ róng, đặc điểm khác nhau giữa các Assembler) Làm sao để chạy những tập tin này trên máy tính? => Linker & Loader LINKER -Thực tế khi lập trình, ta sẽ dùng nhiều file (header / source) liên kết và kèm theo các thư viện có sẵn. -Cần chương trình Linker để liên kết các file sau khi đã biên dịch thành mã máy này (Object file) -Tập tin thực thi (ví dụ: .exe, .bat, .sh) Khi double click vào những tập tin thực thi, cần chương trình tính toán và tải vào memory để CPU xử lý -> Loader
6 bits	5	5	5	5	5	6																									
opcode	hex	rs	dec	rt	rd	shmat funct																									
6 bits	5	5	16	immediate																											
opcode	rs	rt																													
6 bits	26	target address																													
opcode																															
QUY TRÌNH THỰC THI LỆNH (EXECUTE CYCLE) -Tính địa chỉ lệnh, Nạp lệnh, Giải mã lệnh, Tính địa chỉ của toán hạng, Nạp toán hạng. Thực hiện lệnh, Tính địa chỉ của toán hạng chưa kết quả, Ghi kết quả. -Các bước này được lặp đi lặp lại cho tất cả các lệnh tiếp theo -Quy trình này gọi là Instruction cycle – vòng lặp xử lý lệnh	ADDRESSING MODE: Là phương thức định vị trí (địa chỉ hóa) các toán hạng trong kiến trúc MIPS -Có 5 phương pháp chính: +Immediate addressing (Vd: addi \$t0, \$t0, 5) Toán hạng = hàng số 16 bit trong câu lệnh +Register addressing (Vd: add \$t0, \$t0, \$t1) Toán hạng = nội dung thanh ghi +Base addressing (Vd: lw \$t1, 8(\$t0)) Toán hạng = nội dung ô nhớ (địa chỉ ô nhớ = nội dung thanh ghi + hàng số 16 bit trong câu lệnh) +PC-relative addressing (Vd: beq \$t0, \$t1, Label) Toán hạng = địa chỉ đích lệnh nhảy = nội dung thanh ghi PC + hàng số 16 bit trong câu lệnh +Pseudodirect addressing (Vd: j 2500) Toán hạng = địa chỉ đích lệnh nhảy = các bit cao thanh ghi PC + hàng số 26 bit trong câu lệnh	QUÁ TRÌNH THỰC THI FILE TRÊN MÁY High-level language program Program → Compiler → Assembly language program Assembly language program → Assembler → Linker → Computer 1 so TH co the bo: Compiler va Assembler 1 so compiler tao file thuc thi o nhieu nen tang kien truc khac nhau: cross-platform compiler.s																													
QUY TRÌNH NẠP LỆNH (FETCH CYCLE)	<table border="1"><tr><td>MAR ← PC</td><td>MBR ← Memory</td></tr><tr><td>IR ← MBR</td><td>PC ← PC + 1</td></tr><tr><td>Control Unit</td><td></td></tr><tr><td>Thanh ghi PC (Program Counter)</td><td>Lưu địa chỉ (address) của lệnh sắp được nạp</td></tr><tr><td>Thanh ghi MAR (Memory Address Register)</td><td>Lưu địa chỉ (address) sẽ được output ra Address bus</td></tr><tr><td>Thanh ghi MBR (Memory Buffer Register)</td><td>Lưu giá trị (value) sẽ được input / output từ Data bus</td></tr><tr><td>Thanh ghi IR (Instruction Register)</td><td>Lưu mã lệnh sẽ được xử lý tiếp</td></tr></table> • Control Unit di chuyển mã lệnh, cung cấp địa chỉ PC, vào thanh ghi IR • Màn hình, giá trị thanh ghi PC sẽ tăng 1 lường, cung cấp địa chỉ của lệnh vừa được nạp	MAR ← PC	MBR ← Memory	IR ← MBR	PC ← PC + 1	Control Unit		Thanh ghi PC (Program Counter)	Lưu địa chỉ (address) của lệnh sắp được nạp	Thanh ghi MAR (Memory Address Register)	Lưu địa chỉ (address) sẽ được output ra Address bus	Thanh ghi MBR (Memory Buffer Register)	Lưu giá trị (value) sẽ được input / output từ Data bus	Thanh ghi IR (Instruction Register)	Lưu mã lệnh sẽ được xử lý tiếp	QUÁ TRÌNH NẠP LỆNH (FETCH CYCLE)															
MAR ← PC	MBR ← Memory																														
IR ← MBR	PC ← PC + 1																														
Control Unit																															
Thanh ghi PC (Program Counter)	Lưu địa chỉ (address) của lệnh sắp được nạp																														
Thanh ghi MAR (Memory Address Register)	Lưu địa chỉ (address) sẽ được output ra Address bus																														
Thanh ghi MBR (Memory Buffer Register)	Lưu giá trị (value) sẽ được input / output từ Data bus																														
Thanh ghi IR (Instruction Register)	Lưu mã lệnh sẽ được xử lý tiếp																														
logic	 -Chương trình được thu thi trong: CPU -Rام chí la bo phan trung chuyen tu dia cung day len bo vi xu li file .EXE chi chay tren window khong chay tren linux HOẠT ĐỘNG CỦA CPU KHI XỬ LÝ LỆNH -CPU xử lý lệnh qua 2 bước, gọi là chu kỳ lệnh: +Nạp lệnh (Fetch): Di chuyển lệnh từ memory vào thanh ghi (register) trong CPU+Thực thi lệnh (Execute): Giải mã lệnh và thực thi thao tác yêu cầu	<pre>graph LR; Start((Start)) --> Fetch[Fetch next Instruction]; Fetch --> Execute[Execute Instruction]; Execute --> Halt((Halt));</pre>																													

<p>MẠCH TỐ HỢP (TÍCH HỢP)</p> <ul style="list-style-type: none"> -Gồm n ngõ vào (input); m ngõ ra (output) +Mỗi ngõ ra là 1 hàm luận lý của các ngõ vào -Mạch tố hợp không mang tính ghi nhớ; Ngõ ra chỉ phụ thuộc vào Ngõ vào hiện tại, không xét những giá trị trong quá khứ <p>ĐỘ TRỄ MẠCH (PROPAGATION DELAY / GATE DELAY) = Thời điểm tín hiệu ra ổn định - thời điểm tín hiệu vào ổn định</p> <p>THIẾT KẾ- Lập bảng chân trị. - Viết hàm luận lý - Vẽ sơ đồ mạch và thử nghiệm</p> <p>MỘT SỐ MẠCH TỐ HỢP CƠ BẢN</p> <ul style="list-style-type: none"> - Mạch toàn cộng (Full adder) - Mạch giải mã (Decoder)- Mạch mã hoá (Encoder),mạch don,mạch tach <p>SOP – SUM OF PRODUCTS-POS – PRODUCT OF SUM</p>	<p>>Sẽ cần 4 chip nhớ .</p> <p>MỘT SỐ ĐÁNG THỨC CƠ BẢN</p> <table border="1"> <tbody> <tr> <td>$x + 0 = x$</td><td>$x \cdot 0 = 0$</td></tr> <tr> <td>$x + 1 = 1$</td><td>$x \cdot 1 = x$</td></tr> <tr> <td>$x + x = x$</td><td>$x \cdot x = x$</td></tr> <tr> <td>$x + x' = 1$</td><td>$x \cdot x' = 0$</td></tr> <tr> <td>$x + y = y + x$</td><td>$xy = yx$</td></tr> <tr> <td>$x + (y + z) = (x + y) + z$</td><td>$x(yz) = (xy)z$</td></tr> <tr> <td>$x(y + z) = xy + xz$</td><td>$x + yz = (x + y)(x + z)$</td></tr> <tr> <td>$(x + y)' = x' \cdot y'$ (De Morgan)</td><td>$(xy)' = x' + y'$ (De Morgan)</td></tr> <tr> <td>$(x')' = x$</td><td></td></tr> </tbody> </table>	$x + 0 = x$	$x \cdot 0 = 0$	$x + 1 = 1$	$x \cdot 1 = x$	$x + x = x$	$x \cdot x = x$	$x + x' = 1$	$x \cdot x' = 0$	$x + y = y + x$	$xy = yx$	$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	$(x + y)' = x' \cdot y'$ (De Morgan)	$(xy)' = x' + y'$ (De Morgan)	$(x')' = x$		<p>-Các instruction trong MIPS có kích thước ntn? -> có kích thước khác nhau, nhỏ nhất là 32 bit, lớn nhất là 64 bit.</p> <p>-IA-64 là tên gọi của ? -> là tên gọi của 1 kiến trúc bộ lệnh</p> <p>-Phát hành nào không đúng về kiến trúc bộ lệnh (KTBL)</p> <p>+KTBL có thể có nhiều cái đặt khác nhau</p> <p>+Ktbl là một thành phần quan trọng của KT vi xử lí</p> <p>+Ktbl của AMD hoàn toàn khác kiến trúc bộ lệnh của Intel</p> <p>+X86, MIPS là các ktbl</p> <p>-D (do ktbl là phần mềm, còn kt vi xử lí là phần cứng nên không thể là câu b)</p> <p>Thông số FSB ghi kèm các bộ xử lí là loại bus? -> nối bộ vi xử lí và bộ nhớ chính</p> <p>-Hệ số nguyên đang bù 2 có miền giá trị nào? -(2n-1) -> (2n-1 - 1)</p> <p>-X = 0101 0101 phép toán cho kết quả vẫn là X là?</p> <p>->(X AND 0FH) SHR 4) OR (X AND F0h) SHR 4)</p> <p>-Sau khi nhân 2 số nguyên có dấu bằng thuật toán Booth, điều gì sẽ xảy ra? -> Thừa số thứ nhất (M) giữ nguyên giá trị, thừa số thứ 2 (Q) mất giá trị</p> <p>-Trong hệ số chẵn động CXD 32 bit, số dương có thể chuẩn hóa lớn nhất là ? -> 1.[23 bit] * 2127</p> <p>-Rút gọn biểu thức $(x + y + z)^*(x + y' + z')^*(x' + y + z')$ -> $Xy + z'$</p> <p>-Multiplexer có 12 ngõ vào cần bao nhiêu cổng? -> 4 cổng</p> <p>-Ngõ vào băng 1 tại ngõ k thì ngõ ra tạo thành số nhị phân giá trị k, đó là mạch g? -> binary encoder</p> <p>-Trạng thái cảm thấy ra trong mạch nào -> mạch lật RS.</p> <p>-Trong RS, Q(t+1) = Q(t) = 0 thi giá trị 2 ngõ vào ntn? -> S = 0, R = x.</p> <p>-Mục đích của pipeline là? -> tăng throughput</p> <p>-Control hazard xảy ra khi nào? -> 1 lệnh phải chờ kết quả của lệnh khác</p> <p>-Branch - delay-slot là gì? -> lệnh luôn được thực hiện ngay phía sau lệnh rẽ nhánh, bắt chấp kết quả của lệnh ra nhánh</p> <p>-Kiểm tra delayed branch thực thi trong pipeline, nếu comple không tìm được lệnh tiếp thi làm thế nào? -> chèn lệnh no-op vào</p> <p>-Trong data hazard, phương pháp Forwarding không giải quyết được khi gặp tình huống nào? -> Iw->add</p> <p>-Data hazard xảy ra khi -> một lệnh sử dụng kết quả của lệnh trước</p> <p>-Sắp xếp theo thứ tự tăng dần tốc độ truy cập? CD->HDD->RAM->cache</p> <p>-Nhóm phần cứng trong bộ nhớ trong? Register, cache, RAM</p> <p>-Thiết bị tạo bởi công nghệ bán dẫn? -> RAM</p> <p>-DRAM gồm 3 loại, sắp xếp chung theo tốc độ xử lí tăng dần? -> SIMM, DIMM, RIMM</p> <p>-Thiết bị có tốc độ truy xuất nhanh nhất là thanh ghi</p> <p>-Set-associative, set dùng để chỉ cái gì? Chỉ 1 nhóm các line</p> <p>-Associative – mapping tru diem hon so voi direct la? ->xác suất cache hit cao</p>
$x + 0 = x$	$x \cdot 0 = 0$																			
$x + 1 = 1$	$x \cdot 1 = x$																			
$x + x = x$	$x \cdot x = x$																			
$x + x' = 1$	$x \cdot x' = 0$																			
$x + y = y + x$	$xy = yx$																			
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$																			
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$																			
$(x + y)' = x' \cdot y'$ (De Morgan)	$(xy)' = x' + y'$ (De Morgan)																			
$(x')' = x$																				
<p>MẠCH TOÀN CỘNG (FULL ADDER - FA)</p> <ul style="list-style-type: none"> -Mạch tố hợp thực hiện phép cộng số học 3 bit -Gồm 3 ngõ vào (A, B: bit cần cộng – C: bit nhớ) và 2 ngõ ra (kết quả thô từ 0 đến 3 với giá trị 2 và 3 cần 2 bit biểu diễn – S: ngõ tổng, C_0 : ngõ nhớ) <p>MẠCH MÃ HOÁ NHỊ PHÂN (BINARY ENCODER)</p> <p>Có 2^n (hoặc ít hơn) ngõ vào, n ngõ ra</p> <p>-Quy định chỉ có duy nhất một ngõ vào mang giá trị = 1 tại một thời điểm</p> <p>-Nếu ngõ vào = 1 đó là ngõ thứ k thi các ngõ ra tạo thành số nhị phân có giá trị = k</p> <p>MẠCH MÃ HOÁ THEO THÚ TỰ (PRIORITY ENCODER)</p> <ul style="list-style-type: none"> -Các ngõ vào được xem như có độ ưu tiên -Giá trị ngõ ra phụ thuộc vào các ngõ vào có độ ưu tiên cao nhất -Ví dụ: Độ ưu tiên ngõ vào $x_3 > x_2 > x_1 > x_0$ <p>MẠCH GIẢI MÃ (DECODER)</p> <p>Có n ngõ vào, 2^n (hoặc ít hơn) ngõ ra</p> <p>-Quy định chỉ có duy nhất một ngõ ra mang giá trị = 1 tại một thời điểm</p> <p>-Nếu các ngõ vào tạo thành số nhị phân có giá trị = k thi ngõ ra = 1 đó là ngõ thứ k</p> <p>MẠCH ĐÔN (MULTIPLEXER - MUX) Còn gọi là mạch chọn dữ liệu</p> <ul style="list-style-type: none"> -Chọn 1 ngõ trong 2^n ngõ vào để quyết định giá trị của duy nhất 1 ngõ ra -Mạch đòn $2^n - 1$ có 2^n ngõ nhập, 1 ngõ xuất và n ngõ nhập chọn <p>MẠCH TÁCH DEMULTIPLEXER (DEMUX)</p> <ul style="list-style-type: none"> -Chọn 1 ngõ trong 2^n ngõ vào để quyết định giá trị của duy nhất 1 ngõ ra -Mạch DEMUX $1 - 2^n$ có 1 ngõ nhập, 2^n ngõ xuất và n ngõ nhập chọn 	<p>LATENCY: thời gian hoàn thành 1 công việc nhất định THROUGHPUT: số lượng công việc có thể hoàn thành trong 1 khoảng thời gian nhất định</p> <p>Pipeline: là throughput trên toàn bộ cv dc giao Trên tài nguyên k đội, cv thực hiện song song chạy trong 1 ống pipeline</p> <p>Pipelining : lệnh thực hiện gối đầu nhau, tiết kiệm thời gian rồi gửi các công đoạn, tảng tốc độ xử lí lệnh</p> <p>BUỚC THỰC THI LỆNH TRONG MIPS:</p> <ul style="list-style-type: none"> +Ifetch: Instruction Fetch, Increment PC(nạp lệnh) +Ded: Instruction Decode, Read Registers(giải mã lệnh) +Exec: (thực thi) +Mem-ref: Calculate Address(tính toán địa chỉ toán hạng) +Arith-log: Perform Operation(tính toán số học) +Mem: (lưu chuyên với bộ nhớ) <p>Load: Read Data from Memory +Store: Write Data from Memory +WB: Write Data Back to Register (lưu dữ liệu vào thanh ghi)</p> <p>TRỎ NGẠI CỦA PIPELINE:</p> <p>STRUCTURAL HAZARDS: nhiều lệnh dùng chung 1 tài nguyên tại 1 thời điểm</p> <p>DATA HAZARD: lệnh sau dùng dữ liệu kq của lệnh trước</p> <p>CONTROL HAZARD: rẽ nhánh gây ra, lệnh sau đợi kq rẽ nhánh lệnh trước -> Hiện tượng stalls or bubbles</p> <p>Structural hazards #1: single memory-> Giải pháp: tạo 2 bộ nhớ đệm Cache Level 1/CPU (L1 instruction Cache and L1 Data Cache).pần cứng phức tạp hơn...</p> <p>Structural hazards #2: register -> 2 Giải pháp: +RegFile có tốc độ truy cập nhanh (ít hơn 1/ t / ALU/ 1 clock)</p> <p>+Tạo RegFile vs 2 ngõ Read and write độc lập</p> <p>Control hazard: rẽ nhánh -> Giải pháp: trì hoãn stall đến lúc điều kiện rẽ nhánh được thực hiện-> chèn lệnh rác(or hoãn việc nạp lệnh sang lệnh kệ(trong 2 chu kỳ clock))</p> <p>+optimization 1 (sự tối ưu hóa): chèn thêm pép so sánh rẽ nhánh đặc biệt tại stage 2(decode)-> quyết định giá trị mới cho thanh ghi (lợi ích: chỉ có 1 lệnh không cần thiết được nạp -> cần 1 no-op là đủ)</p> <p>+optimization 2: tái định nghĩa rẽ nhánh: khi thực thi rẽ nhánh, một lệnh ngay sau lệnh rẽ nhánh sẽ dc thực thi (brand-delay slot: xâu nhất: đặt 1 lệnh no-op vào trong br. Tốt nhất: tìm dc 1 lệnh trc lệnh rẽ nhánh, k ảnh hưởng đến chương trình.)</p> <p>Giải pháp Data hazard:</p> <ul style="list-style-type: none"> +Forwarding: Trì hoãn lệnh sub lại sau đó dùng forwarding.+Load: vị trí lệnh sau 1 load gọi là load delay slot.+ Lệnh dùng kq của load, hardware interlock có thể se hoãn nó đúng 1 chu kỳ clock (lệnh k liên quan-> hoãn lệnh đó). 	<p>-> một lệnh sử dụng kết quả của lệnh trước</p> <p>-Sắp xếp theo thứ tự tăng dần tốc độ truy cập? CD->HDD->RAM->cache</p> <p>-Nhóm phần cứng trong bộ nhớ trong? Register, cache, RAM</p> <p>-Thiết bị tạo bởi công nghệ bán dẫn? -> RAM</p> <p>-DRAM gồm 3 loại, sắp xếp chung theo tốc độ xử lí tăng dần? -> SIMM, DIMM, RIMM</p> <p>-Thiết bị có tốc độ truy xuất nhanh nhất là thanh ghi</p> <p>-Set-associative, set dùng để chỉ cái gì? Chỉ 1 nhóm các line</p> <p>-Associative – mapping tru diem hon so voi direct la? ->xác suất cache hit cao</p>																		
<p>5 THANH PAN CƠ BẢN CỦA MÁY TINH: processor (control, datapath), memory, devices(input,output).</p> <p>-Bộ vi xử lí (CPU): datapath (registers,ALU), control unit, stalling: CPU= registers, ALU, control unit, internal bus;</p> <p>-Lệnh muốn thực thi: gửi địa chỉ lệnh chứa trong PC _bó nhớ _XD toàn hàng_đọc thanh ghi chứa toàn hàng _địa chỉ tương ứng (2 buoc).Cac bc tt pt tung lehnh khac nhau.</p> <p>-Xây dựng đường đi dữ liệu (Datapath): xu kiến trúc phần tử cần thiết cho cầu lệnh, xây dựng phân khúc, xu hoán chính datapath cho cầu lệnh.</p> <p>-Tập thank ghi: 3 ngõ nhận dữ liệu, 1 ngõ ghi, 2 ngõ đọc, 1 tín hiệu điều khiển ghi</p> <p>-Đơn vị số học: 2 ngõ toan hoàn hàng, 1 ngõ ra, 1 bit zero, 1 tín hiệu điều khiển (4 bit)</p> <p>-Thêm 2 thanh pan cơ bản: bộ nhớ dữ liệu 1 ngõ nhận địa chỉ ô nhớ, 1 ngõ nhận dữ liệu cần ghi, 1 ngõ dữ liệu đọc, 2 tín hiệu điều khiển doc/ghi, bộ nhớ rộng dầu(1 nhập, 1 ra)> L1 format</p> <p>-Datapath cho R_Format lw,sw,beq</p> <p>Tín hiệu dk ALU: 00000(and), 00011(or), 0010(add), 0110(sub),0111(st),1100(not)</p> <p>-CPU đơn chí thê k dc vì thời gian thiện lệnh khác nhau, trùng lặp pnt</p> <p>-CPU đa chí thi thi 1 câu lệnh thành nhiều chu kỳ clock, thời gian thực thi theo giàn dò trạng thái, dùng 1 bộ nhớ chung cho các câu lệnh, thêm vào 1 số thanh ghi chứa dữ liệu/ kq trung gian</p> <p>-Tap lenh MIPS thu gon co cac buoc thuc thi giang kha nheu diem, khac chu yeu o cac buoc thuc thi cuoi cua cau lenh.</p>	<p>+Assocciative mapping: moi Block co the nap vao bat ky Line nao cua Cache, Dc bo nho om: Word, Tag</p> <p>VD: bộ nhớ chính = 4GB -> N=32 bit -1 Line= 1Block= 32 byte= 2^5 -> W= 5 bit -T=N-W=32-5 =27 bit = M</p> <p>+Set associative mapping: Cache duoc chia thanh cac tap</p> <p>Vd: Bộ nhớ chính= 4GB -> N=32 bit. -Dung luong Cache=256 KB = 2^{18}byte Tag xd Block nao trong bo nho chinh dang nam o Line do.</p> <p>1Line=1block=32 byte = 2^5 byte -> W= 5 bit. -Số line trong Cache= $2^{18} = 2^{13}$Line->L=13 bit</p> <p>Một set trong cache có 4 Line =2^2Line. -Số set trong Cache = $\frac{2^{13}}{2^2} = 2^{11}$set-> S= 11 bit</p> <p>$T = N + (S + W) = 32 - (11 + 5) = 16$ bit. -XD (W,S,T) theo kiểu 4-way associative mapping</p> <p>-Tham số ảnh hưởng bộ nhớ cache: block size,cache size</p> <p>-Thuật toán thay thế: Random, FIFO(thay thế line nằm lâu nhất trong Cache), LFU (Line có số lần truy cập ít cùng 1 thời điểm), LRU (Line có thời gian lâu nhất k dc tham chiếu tới, tối ưu nhất)</p> <p>-Write Policy: +1 line bị thay đổi trong cache, sẽ thực hiện theo cách ghi lên lại RAM : write through, write back(line bị thay thế)+nhiều processor chia sẻ RAM, mỗi processor có cache riêng: bus watching with WT(loại bỏ line khi thay đổi trong 1 cache khác), hardware transparency(tự động cập nhật các cache khác khi line bị 1 cache thay đổi), noncacheable share memory (pân bộ nhớ không dung chung sẽ k dc đưa vào cache)</p> <p>-Số lượng và loại cache: +Mức cache : L1, L2..... + Co the chung cho ca data va instruction hoc rieng.</p> <p>+Mức thấp : onchip, cao: offchip truy cập qua external bus or bus đanh riêng</p> <p>-Cache trên các bộ xử lý Intel: +80486: 8 KB cach L1 trên chip (on-chip)+Pentium: có 2 level cache L1 trên chip +cache lệnh: 8KB +cache dữ liệu: 8KB+ Pentium 4(2000): có 2 level cache L1 và L2 trên chip +cache L1: 2 cache, mỗi cache 8KB; kích thước Line = 64byte; 4-way associative mapping +cache L2: 256KB; kích thước Line = 128byte; 8-way associative mapping</p>	<p>HỆ THỐNG NHẬP XUẤT</p> <p>I/O devices : 2 cách tổ chức:</p> <p>+port-mapped I/O: có thể cần instruction riêng cho I/O, sd không gian địa chỉ riêng cho các thiết bị, mỗi thiết bị được gắn 1 port</p> <p>+memory-mapped I/O: k cần thêm instruction riêng, dùng chung gian bộ nhớ, mỗi thiết bị được cấp một vùng địa chỉ, làm việc với thiết bị giống như làm việc với bộ nhớ.</p> <p>Giao tiếp với thiết bị: control register, data register</p> <p>>Có chép:</p> <p>Polling: CPU chủ động kt trạng thái của thiết bị</p> <p>Interrupt-driven: thiết bị chủ động thông báo trạng thái vs CPU</p> <p>DMA: giao tiếp không qua CPU</p> <p>Interrupt: 1 trong 4 loại exception (interrupt, trap, fault, abort)</p> <p>Interrupt service handler: cách dc đến được đoạn code xử lí interrupt : centralized dispatch, vectored dispatch</p>																		
<p>BỘ NHỚ:</p> <p>Từ trái sang phải: dung lượng tăng dần_tốc độ giảm dần_giá thành 1 bit giảm dần->Bộ nhớ trong: bộ nhớ Cache L1_Cache L2_bộ nhớ chính</p> <p>Phân loại: +PP truy cập: tuân tự (băng tú), trực tiếp(các loại đĩa), ngẫu nhiên (bộ nhớ bẩn đầm RAM,ROM), liên kết(cache)+Kiểu vật lí: bộ nhớ bẩn đầm, bộ nhớ từ (HDD, FDD), bộ nhớ quang (CD-ROM,DVD)+Bộ nhớ ngoài: băng tú(magnetic tape), đĩa tú (magnetic disk),đĩa quang (optical disk), flash disk</p> <p>-Hệ thống nhớ lưu trữ lớn RAID (redundant array of inexpensive disk)</p> <p>+Xem thêm 1 ổ logic duy nhất có dung lượng lớn+Dữ liệu được lưu trữ trên các ổ đĩa vật lí -> truy cập song song +SD dung lượng dư thừa để lưu trữ thông tin, khởi pục thông tin khi đĩa bị hỏng-> an toàn thông tin-Có 7 loại RAID (0-6).</p> <p>-Bộ nhớ trong: +Bộ nhớ chính: dạng các module nhớ DRAM(bit lưu trữ trên ty điện)-cần mạch refresh,cấu trúc đơn giản, dung lượng lớn, speed chậm, re tiền, dung lạm bộ nhớ chính)</p> <p>*Chương trình đang thực hiện, dữ liệu đang thao tác, tồn tại trên mọi hệ thống, ngắn nhớ dc đánh địa chỉ trực tiếp bởi CPU, dung lượng < kgian địa chỉ bộ nhớ mà CPU quản lý, công nghệ lưu trữ DRAM</p> <p>*Phân loại Dram: SIMM(cù, chậm); RIMM(mới, n nhất)</p> <p>+Bộ nhớ đệm: tích hợp trên chip của CPU, sd công nghệ lưu trữ SRAM (bit lưu trữ bằng các flip flop-> thông tin ôn định, cấu trúc phức tạp, dung lượng chip nhỏ, speed nhanh, đắt tiền, dung lạm bộ nhớ Cache)</p> <p>-Khi đọc 1 ô nhớ từ bộ nhớ, nếu chưa có cache miss: chép ô nhớ đó và 1 số ô nhớ lân cận từ bộ nhớ chính vào cache; nếu có cache hit: đọc từ cache, k cần truy xuất bộ nhớ chính. -Cache là bản copy một phần bộ nhớ chính, dùng công nghệ SRAM, truy xuất cao hơn bộ nhớ chính. -Nguyên lí cơ sở khi truy xuất: temporal locality (cục bộ về thời gian), spatial locality (cục bộ về không gian). CPU (truy xuất từng byte/word)> cache(truy xuất từng block) -> RAM</p> <p>-PP ánh xạ: +Direct mapping (ánh xạ trực tiếp) KBNc=4 GB</p> <p>Vd: bộ nhớ chính=4GB=2^{32}byte-> N=32 bit kt 1 line=1block=32 byte. (W,S,T)</p> <p>dung luong cache=256KB=2^{18}byte -> dung 18 bit đánh địa chỉ o nhớ trong Cache. -1line=1 block=32 byte=2^5 byte->W=5 bit. -Số Line trong Cache= $2^{18} = 2^{13}$Line -> L=13 bit.</p> <p>Mot Set trong Cache co 4 line. (to chuc theo kieu 4-way associative mapping), --->trong Cache=2¹³/2²=2¹¹. S=11,Tag=T-(S+W)=32-(11+5)=16 bit danh gia: bo so sanh don gian, xac suat cache hut thap. Bi xung dot thi ca 2 o duoc luu o Line thu 0</p>																				

1.Chuyển đổi cơ số:

- a.123dec=01111101bin
- b.BE hex=190d=(11*16+14)
- MSB: Bit trái nhất là bit có giá trị (nặng) nhất.
- LSB: Bit phải nhất là bit có giá trị (nhẹ) nhất.

2. Số nguyên có dấu:

- Dấu lượng: 1 byte 8 bit biểu diễn các số từ -127 đến 127.
 - +0: số dương, 1: số âm.
 - Bù 1: đáo bit các bit làm độ lớn.
 - Bù 2:= bù 1+1 (-128→127)
 - Số quá k: +N. (-128→127)
- 3.Tính gt không dấu và có dấu:**
- Ví dụ: 1100 1100 1111 0000
- Số nguyên không dấu ? 52464
 - Số nguyên có dấu ?
 - + Bit MSB = 1 do đó số này là số âm + áp dụng ct: -13072

Name	Abbr	IEC	SI
Kibi/Kilo	Ki/K	2^{10}	10^3
Mebi/Mega	Mi/M	2^{20}	10^6
Gibi/Giga	Gi/G	2^{30}	10^9
Tebi/Tera	Ti/T	2^{60}	10^{12}
Pebi/Peta	Pi/P	2^{50}	10^{15}
Exbi/Exa	Ei/E	2^{60}	10^{18}
Zebi/Zetta	Zi/Z	2^{70}	10^{24}

-IEC: dùng trong đo lường các phần mềm trong lập trình.

-SI: Hiện nay chỉ có các nhà sản xuất đĩa cứng và viễn thông mới dùng.

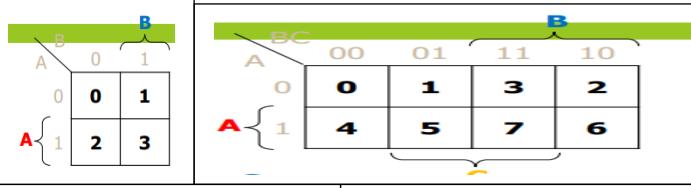
$$+30 \text{ GB} \rightarrow 30 * 109 \sim 28 * 230 \text{ bytes}$$

$$+1 \text{ Mbit/s} \approx 106 \text{ b/s}$$

Chú ý: khi nói “kilobyte là 1000 bytes theo chuẩn SI, 1024 bytes là kibibyte theo chuẩn IEC.

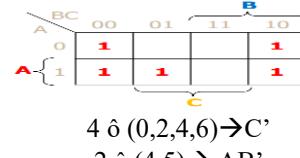
RS Flip-Flop: thêm 1 tín hiệu ngõ vào kích hoạt “Enabled” (thường là tín hiệu xung đồng hồ Clock - C) để điều khiển mạch.+Enabled = 1 (Positive Clock Edge): mạch hoạt động như mạch latches + Enabled = 0 (Negative Clock Edge): mạch bị vô hiệu hóa → Q giữ nguyên giá trị à Q(t+1) = Q(t) → Chỉ khi tín hiệu Enabled đổi từ 0 sang 1 (positive edge triggered), ngõ ra mới có thể bị ảnh hưởng, nếu không thì không thể thay đổi bất chấp giá trị của S và R.

E	S	R	$Q = Q(t+1)$	Q'	Ý nghĩa
0	x	x	$Q(t)$	$(Q(t))'$	Không đổi
1	0	0	$Q(t)$	$(Q(t))'$	Không đổi
1	0	1	1	0	= 0
1	1	0	1	0	= 1
1	1	1	undefined	undefined	Không xác định



Theo SOP:

$$F(A, B, C) = \sum(0, 2, 4, 5, 6)$$



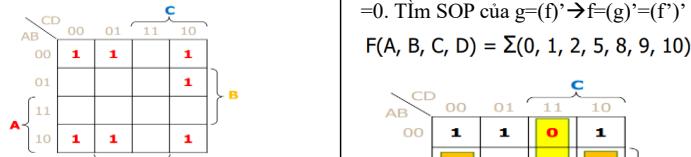
$$4 ô (0,2,4,6) \rightarrow C'$$

$$2 ô (4,5) \rightarrow AB'$$

$$\rightarrow F(A,B,C) = C' + AB'$$

Theo SOP:

$$F(A, B, C, D) = \sum(0, 1, 2, 6, 8, 9, 10)$$



$$4 ô (0,1,8,9) \rightarrow B'C'$$

$$4 ô (0,2,8,10) \rightarrow B'D'$$

$$2 ô (2,6) \rightarrow A'CD'$$

$$F(A, B, C) = B'D' + B'C' + A'CD'$$

-1 số trường hợp không cần quan tâm đến giá trị ngõ ra của 1 số bộ tri nào đó (1 hay 0 đều được)

-Trong bản đồ ta sẽ ghi tương ứng những ô đó là x

-x có thể gom nhóm với các ô liền kề nhằm đơn giản hóa

-Lưu ý: Không gom nhóm bao gồm toàn những ô có giá trị x.

$$F(A, B, C) = \sum(0, 2, 6)$$

$$d(A, B, C) = \sum(1, 3, 5)$$

Vị trí	A	B	C	D
0	0	0	0	0
1	0	0	1	x
2	0	1	0	1
3	0	1	1	x
4	1	0	0	0
5	1	0	1	x
6	1	1	0	1
7	1	1	1	0



$$\rightarrow F(A,B,C) = A' + BC$$

- Phân loại mạch latches: Latch,Flip-Flop.

+ Latch: ngõ ra thay đổi trạng thái khi ngõ vào thay đổi giá trị, Độ trễ mạch (delayed gate) giá trị mới của ngõ ra được xác định bằng độ trễ giữa ngõ vào và ngõ ra, Được sử dụng như 1 thành phần nhỏ của mạch tuần tự bắt đồng bộ.

+Flip-Flop: Bên cạnh những ngõ vào thông thường thì luôn có 1 ngõ vào kích hoạt (trigger input), gọi là clock. Trạng thái của ngõ ra chỉ có thể thay đổi khi ngõ vào kích hoạt (clock) thay đổi xung đồng hồ (clock pulse) của nó (0 → 1 hoặc 1 → 0). Được sử dụng như 1 thành phần nhỏ của mạch tuần tự đồng bộ.

- SR Latch: Có 2 ngõ vào: S: Đặt , R: Khởi động .Có 2 ngõ ra Q và Q'. Trạng thái ngõ ra Qnext = Q(t+1) phụ thuộc vào trạng thái ngõ vào S, R và tình trạng hiện tại của mạch Qcurrent = Q(t).

S	R	$Q = Q(t+1)$	Q'	Ý nghĩa
0	0	$Q(t)$	$(Q(t))'$	Không đổi
0	1	0	1	= 0
1	0	1	0	= 1
1	1	undefined	undefined	Không xác định

;Tinh thuong mov eax,[numa] idiv dword[numb mov [kq],eax _KTNT.TangDem: inc ecx	;khoi tao mov ecx,0 ; dem mov esi,1; i = _KTNT.Lap: xor edx,edx mov eax,[ebp + idiv esi cmp edx,0 je _KTNT.TangDe jmp KTNT.Tangi
--	---

*THUẬT TOÁN NHÂN:

Phép nhân M x Q với Q có n bit
Khởi tạo: [C, A] = 0;

k = n

Lặp khi k > 0

{
 Nếu bit cuối của Q = 1 thì:
 Lấy (A + M) → [C, A]
 SHR [C, A, Q] ;
 k = k - 1;
 }
TTN CẢI TIẾN:

Khởi tạo: A = 0; k = n; Q-1 = 0 (thêm 1 bit = 0 vào cuối Q)
Lặp khi k > 0

{
 Nếu 2 bit cuối của Q0Q-1
 = 10 thì A - M → A
 = 01 thì A + M → A
 = 00, 11 → A không đổi
}
Shift right [A, Q, Q-1]
k = k - 1
} Kết quả: [A, Q]
; thanh tuc
 mov eax,1 ; T = 1
 mov esi,1; i = 1
_TinhMu.Lap:
 imul dword[ebp + 8]; T = T * x
 inc esi
 ;kiem tra i <= n thi Lap
 cmp esi, [ebp + 12]
 jle _TinhMu.Lap

*THUẬT TOÁN NHÂN: Q/M

Khởi tạo: A = n bit 0 nếu Q > 0;
A = n bit 1 nếu Q < 0; k = n.

Lặp khi k > 0

{
 SHL [A, Q]
 A - M → A
 # Nếu A < 0: Q0 = 0
 và A + M → A
 # Ngược lại: Q0 = 1
 k = k - 1
}
Kết quả: Q là thương, A là số dư

Mạch số: Là thiết bị điện tử hoạt động với 2 mức điện áp: Cao: 1, Thấp: 0.
- Được xây dựng từ những thành phần cơ bản là **công luận lý** (là thiết bị điện tử gồm 1 / nhiều input - 1 tín hiệu đầu ra (output)) +Tùy thuộc vào cách xử lý của hàm F sẽ tạo ra nhiều loại công luận lý
- L/kien cơ bản để tạo ra mạch số là **transistor**.

-D Flip-Flop:

E	D	$Q = Q(t+1)$	Q'	Ý nghĩa
0	x	$Q(t)$	$(Q(t))'$	Không đổi
1	0	0	1	= 0
1	1	1	0	= 1
1	1	undefined	undefined	Không xác định

_KTNT.Tangi:

```
inc esi  
;kiem tra i <= n thi Lap  
cmp esi, [ebp + 8]  
jle _KTNT.Lap  
; Kiem tra dem = 2 return 1  
cmp ecx, 2  
je _KTNT.Return1  
mov eax,0  
jmp _KTNT.KetThuc
```

JK Flip-Flop: R = S = 1. Nguyên tắc: J = S, K = R. Nếu J = K = 1 thì khi đó với 1 chuỗi tiếp của tín hiệu xung đồng hồ sẽ chuyển tín hiệu ngõ ra Q sang trạng thái bù Q'.

J	K	Q = Q(t+1)	Q'	Ý nghĩa
0	0	Q(t)	(Q(t))'	Không đổi
0	1	0	1	= 0
1	0	1	0	= 1
1	1	Q'(t)	Q(t)	Đảo bit

Mạch lật T: kết hợp 2 ngõ vào J, K thành duy nhất 1 ngõ vào T (T = J = K). T = 0: Q(t + 1) = Q(t) . T = 1: Q(t + 1) = (Q(t))'

T	Q = Q(t+1)	Q'	Ý nghĩa
0	Q(t)	(Q(t))'	Không đổi
1	(Q(t))'	Q(t)	Đảo bit

Bảng Kích thích mạch tuần tự:

Mạch lật RS / SR			
Q(t)	Q(t+1)	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Cú pháp tổng quát lệnh Mips:

<ten-lenh> <r1>, <r2>, <r3>
- r1: thanh ghi chưa kết quả
- r2: thanh ghi
- r3: thanh ghi hoặc hằng số

Rd: thanh ghi đích
Rs,Rt: thanh ghi nguồn

Cách khai báo biến

tên biến: kiểu_lưu_trữ giá_trí
Các kiểu lưu trữ hỗ trợ: .word, .byte, .ascii, .space

VD: a: .word 3 #int 4 byte k/t=3
v: .byte 'a', 'b'
v: .space 40 #cấp 40 byte bộ nhớ, chưa được khởi tạo.

Mạch lật JK			
Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Mạch lật D		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Mạch lật T		
Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Cấu trúc cơ bản của 1 chương trình hợp ngữ trên MIPS:

.data # khai báo các data label (VD: biến)
sau chỉ thị này

label1:<kiểu lưu trữ><gti khai tao>

label2:<kiểu lưu trữ><gti khai tao>

.text # viết các lệnh sau chỉ thị này

.globl<các text label toàn cục, có thể truy xuất từ file khác>

.globl main # Đây là text label toàn cục bắt buộc của program

main: # điểm text label bắt đầu của program

VD: Hello.asm:

```

.data          # data segment
str: .asciz "Hello asm !"
.text          # text segment
.globl main
main:          # starting point of program
    addi $v0, $0, 4      # $v0 = 0 + 4 = 4 → print str syscall
    la $a0, str          # $a0 = address(str)
    syscall              # execute the system call

```

-Có 4 thao tác chính: Phép toán số học, Di chuyển dữ liệu, Thao tác luận lý, Rẽ nhánh.

1. Phép toán số học:

Opt opr, op1, opr2

+ Cộng có dấu: add \$s0,\$s1,\$s2

+ Cộng k dấu: addu \$s0,\$s1,\$s2

+\$s2 có thể đóng vai trò hằng số.

+ Gán: add \$s0,\$s1,\$zero. (a=b)

+ Nhân: mult \$s0,\$s1 ; mflo \$s0 (\$s0=lo); Mfhi \$s0.

+ Chia: div \$s0,\$s1 (thương). Shi: số dư.

MIPS cung cấp 2 loại lệnh số học:

+ Phát hiện tròn số: add, addi, sub

+ K p/h tròn: addu, addiu, subu(C)

2. Di chuyển dữ liệu: lw, sw

+ move Rd,Rs //Rd=Rs: di chuyển giữa 2 thanh ghi

A[12]=h-A[8], A trong \$s3,g h trong \$s1 \$s2

Lw \$t0,32(\$s3); sub \$t0,\$s2,\$t0;

Sw \$t0,48(\$s3)

3 cờ điều khiển : T hoặc TF (trap flag): cờ bẫy, TF=1 khi CPU làm việc ở chế độ chạy từng lệnh; I hoặc IF (Interrupt enable flag): cờ cho phép ngắn, IF=1 thì CPU sẽ cho phép các yêu cầu ngắn (ngắt che được) được tác động (Các lệnh: STI, CLI); D hoặc DF (direction flag): cờ hướng, DF=1 khi CPU làm việc với chuỗi ký tự theo thứ tự từ phải sang trái (lệnh STD, CLD)

Thanh ghi: Là đơn vị lưu trữ data duy nhất trong CPU

Trong kiến trúc MIPS: Có tổng cộng 32 thanh ghi đánh số từ \$0 → \$31. Mỗi thanh ghi có kích thước cố định 32 bit/giới hạn bởi k/n tính toán của chip xl, nhóm 32 bit là word).

Thanh ghi đa năng:

Số	Tên	Ý nghĩa
\$0	\$zero	Hàng số 0
\$1	\$at	Assembler Temporary
\$2-\$3	\$v0-\$v1	Giá trị trả về của hàm hoặc biến thức
\$4-\$7	\$a0-\$a3	Các tham số của hàm
\$8-\$15	\$t0-\$t7	Thanh ghi tạm (không giữ giá trị trong quá trình gọi hàm)
\$16-\$23	\$s0-\$s7	Thanh ghi lưu trữ (giữ giá trị trong suốt quá trình gọi hàm)
\$24-\$25	\$t8-\$t9	Thanh ghi tạm
\$26-\$27	\$k0-\$k1	Dự trữ cho nhân HDH
\$28	\$gp	Con trỏ toàn cục (global pointer)
\$29	\$sp	Con trỏ stack
\$30	\$fp	Con trỏ frame
\$31	\$ra	Địa chỉ trả về

Thanh ghi HI và LO: Thao tác nhân của MIPS có kết quả chứa trong 2 thanh ghi HI và LO. Bit 0-31 thuộc LO và 32-63 thuộc HI.

Thanh ghi dấu phẩy động: sử dụng 32 thanh ghi dấu phẩy động để biểu diễn độ chính xác đơn của số thực. Các thanh ghi này có tên là : \$f0 – \$f31.

Để biểu diễn độ chính xác kép (double precision) thì MIPS sử dụng sự ghép đôi của 2 thanh ghi có độ chính xác đơn.

Dịch vụ	Giá trị trong \$v0	Đối số	Kết quả
print_int	1	\$a0 = integer	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer (trong \$v0)
read_float	6		float (trong \$f0)
read_double	7		double (trong \$f0)
read_string	8	\$a0 = buffer, \$a1 = length	
sbrk	9	\$a0 = amount	address (trong \$v0)
exit	10		
print_character	11	\$a0 = char	
read character	12		char (trong \$v0)

3. Thao tác luận lý:

+and, or, nor: Toán hạng nguồn

thứ 2 (opr2) phải là thanh ghi.

+andi, ori: Toán hạng nguồn thứ 2 (opr2) là hằng số.

VD: sll \$s1,\$s2,2 #dịch trái luận

lý \$s2 2 bit.

Loop:

sll \$t1,\$s3,2 # \$t1 = i * 22

add \$t1,\$t1,\$s5 # \$t1 = adrA[i]

lw \$t1,0(\$t1) # \$t1 = A[i]

add \$s1,\$s1,\$t1 # g = g + A[i]

add \$s3,\$s3,\$s4 # i = i + j

bne \$s3,\$s2, Loop

if (i != i) goto Label

j label

Nhập không điều kiện đến nhãn 'label'

jal label

Lưu địa chỉ trả về vào \$ra và nhảy đến nhãn 'label' (dùng khi gọi hàm)

jr Rs

Nhảy đến địa chỉ trong thanh ghi Rs (dùng để trả về từ lời gọi hàm)

Thủ tục: Đôi số \$a0, \$a1, \$a2, \$a3; Kết quả trả về

\$v0, \$v1; Biến cục bộ \$s0, \$s1, ..., \$s7. Địa chỉ trả về

về \$ra. Nhảy đến jal về bằng jr \$ra. Tuân thủ theo nt

sử dụng thanh ghi.

Đầu thủ tục

entry_label:

addi \$sp,\$sp,-framesize # khai báo kích thước cho stack

sw \$ra, framesize-4(\$sp) # cát địa chỉ trả về của thủ tục trong \$ra vào ngăn xếp

Lưu tạm các thanh ghi khác (nếu cần)

Thân thủ tục ...

(có thể gọi các thủ tục khác...)

Cuối thủ tục

Phục hồi các thanh ghi khác (nếu cần)

lw \$ra, framesize-4(\$sp) # lấy địa chỉ trả về ra \$ra

addi \$sp,\$sp,framesize

jr \$ra

-\$s0, \$s1, ..., \$s7: khôi phục lại nếu thay đổi.

-\$sp: khôi phục nếu thay đổi, Thanh ghi con trả stack

phải có giá trị không đổi trước và sau jal.

• Thanh ghi chứa AX : kết quả của các phép

tính. Kq 8 bit được chứa trong AL • Thanh ghi

cơ sở BX (base): chứa địa chỉ cơ sở. • Thanh

ghi đếm CX chứa số lần lặp trong các lệnh lặp.

CLD được dùng để chừa địa chỉ lặp qua

trong các lệnh dịch và quay thanh ghi • Thanh

ghi dữ liệu DX (data): cùng AX chứa dữ liệu

trong các phép tính nhân chia số 16 bit. DX còn

được dùng để chừa địa chỉ công trong các lệnh

vào ra dữ liệu trực tiếp (IN/OUT)

Cấu trúc của một chương trình hợp ngữ

section .data

<Khai báo dữ liệu (kiểu tĩnh)>

section .bss

<Khai báo dữ liệu (kiểu động)>

_mainCRTStartup: ; Nhập bắt đầu chương trình, có thể thay đổi

; tùy thuộc vào loại project tạo trên Visual C

< Các lệnh thực thi >

section .text

global CMAIN

;đầu thu tục

global _TinhMu

_TinhMu:

; backup ebp

push ebp

mov ebp,esp

;xoa stack

sub esp,32

;backup thanh ghi neu can

push esi

;tham so 1 : x = [ebp + 8]

;tham so 2 : n = [ebp + 12]

ret

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;