

データあたりのコスト = ウエーハあたりのコスト	ウエーハあたりのターン数 × 歩留まり	ウエーハあたりのターン数 ≈ $\frac{\text{ウエーハ面積}}{\text{ターン面積}}$	歩留まり = $\frac{1}{1 + (\text{単位面積の欠陥数} \times \text{ターンの面積}/2)}$				
改善率(%) = $\left\{ \left(\frac{\text{コストB}}{\text{コストA}} \right) - 1 \right\} \times 100$	$10^{18} = \text{exa(E)} / 10^{15} = \text{pet(a)} / 10^{12} = \text{tera(T)} / 10^9 = \text{giga(G)} / 10^6 = \text{mega(M)} / 10^3 = \text{kilo(k)}$	$10^{-18} = \text{atto(a)} / 10^{-15} = \text{femto(f)} / 10^{-12} = \text{pico(p)} / 10^{-9} = \text{nano(n)} / 10^{-6} = \text{micro(\mu)} / 10^{-3} = \text{milli(m)}$	$10^6 = \text{million} / 10^9 = \text{billion} / 10^{12} = \text{trillion} / 10^{15}$				
CPU 実行時間	レジスタ番号: レジスタ名 = {0:\$zero, 1:\$at, 2:\$v0, 3:\$v1, 4:\$a0, 5:\$a1, 6:\$a2, 7:\$a3, 8:\$t0, 9:\$t1, 10:\$t2, 11:\$t3, 12:\$t4, 13:\$t5, 14:\$t6, 15:\$t7, 16:\$s0, 17:\$s1, 18:\$s2, 19:\$s3, 20:\$s4, 21:\$s5, 22:\$s6, 23:\$s7, 24:\$t8, 25:\$t9, 26:\$k0, 27:\$k1, 28:\$gp, 29:\$sp, 30:\$fp} 1byte = 8bit + 1	クロック周同期 = クロックサイクル時間 = $\frac{1}{\text{クロック周波数}}$	クロック周波数 = $\frac{1}{\text{クロックサイクル時間} (= \text{クロック周期})}$				
CPU クロックサイクル数 × クロックサイクル時間	CPU クロックサイクル数	MIPS = $\frac{\text{実行命令数}}{\text{実行時間} \times 10^6}$	MIPS = 1秒間に実行できる100万単位の命令数				
クロック周波数	実行命令数 × CPI (命令あたりの平均クロックサイクル数)	性能 A = $\frac{\text{実行時間 B}}{\text{実行時間 A}}$	[ステージ] IF: 命令フッテ, ID: 命令コードとレジスタフェッチ, EX: 命令実行/アドレス生成, MEM: データ・メモリアクセス, WB: 書き込み				
クロック周波数	実行命令数 × $\frac{\text{クロックサイクル数}}{\text{命令数}} \times \frac{\text{秒数}}{\text{クロックサイクル数}}$	性能 B = $\frac{\text{実行時間 A}}{\text{実行時間 B}} = n(A \text{が } B \text{より } n \text{倍速い})$					
実行命令数 × CPI = $\frac{\text{実行時間}}{\text{実行命令数}} = \frac{\text{クロックサイクル数}}{\text{実行命令数}}$	命令クラス A, B, C があるとする	平均 CPI = $\frac{CPI_A \times \text{実行命令数} A + CPI_B \times \text{実行命令数} B + CPI_C \times \text{実行命令数} C}{\text{実行命令数} A + \text{実行命令数} B + \text{実行命令数} C}$					
実行時間 = $\frac{\text{CPI}}{\text{クロック周波数}} \times \text{実行命令数}$	(R-format) 6 5 5 5 5 6	[I-format] 6 5 5 5 5 6	* 4倍速いのに注意				
クロック周波数 = $\frac{\text{実行命令数}}{\text{実行時間}} \times CPI$	op rs rt rd func + shamt	op rs rt rd immediate					
命令数 = $\frac{\text{クロック周波数}}{CPI} \times \text{実行時間}$	add 0 2 3 1 0 32 add \$1,\$2,\$3 addu 33 addu \$1,\$2,\$3 sub 34 subu 35 and 36 or 37 not 39 slt 42 sltu 43 sll 0 0 2 1 0 0 sll \$1,\$2,10 srl 0 0 2 1 0 02 sr \$1,\$2,10 jrh 0 31 0 0 0 8 jrh \$31	beq 4 1 2 25(offset) beq \$1,\$2,25 bne 5 1 2 25(offset) bne \$1,\$2,25 addi 8 2 1 100 addi \$1,\$2,100 addiu 9 andi 12 ori 13 slt 10 sltu 11 lui 15 0 1 100 lui \$1,100 lw 35 2 1 100(offset) lw \$1,100(\$2) sw 43 2 1 100(offset) sw \$1,100(\$2)	-4倍 1100 -8 11000				
I-format] 6 5 5 5 5 6, OP address j 2 10000 j 10000 jal 3 10000 jal 10000	※ 1/4レジスタに注意 名前: {0}:{1}						
RegDst: {書込みレジスタのアドレス - レジスタ番号がドセントルード(ビット20-16)から得られる}, {書込みレジスタのアドレス - レジスタ番号がトドルード(ビット11-1)から得られる}, RegWrite: {なし}, {書込みレジスタ入力に複数オペランドレジスタに書込みメモリ入力の値が書き込まれる}, ALUSrc: {ALUの第2オペランドがALUの第2出力(読み出しデータ2)から得られる}, {ALUの第2オペランドが命令の下位16ビットで符号拡張したものになる}, PCSrc: {PC+4で計算した算器の出力によってPCが選択される}, {分歧先を計算した加算器の出力によってPCが選択される}, MemRead: {なし}, {読み出しアドレスによって定めたデータ・メモリの内容が読み出しデータ出力上に流れ来る}, MemWrite: {なし}, {書込みアドレスによって指定されたアドレスにあるデータ・メモリの内が書込みデータ入力の値によって書きかわされる}, MemReg: {レジスタの書込みメモリ入力へ渡される値がALUから得られる}, {レジスタの書込みメモリ入力へ渡される値がデータ・メモリから得られる}							
put3	R w sw beg addi andi ori set less than immediate	マルチクロックサブルートソース 描写	1a. EX/MEM. Register Rd = ID/EX, Register Rs; 1b. EX/MEM. Register Rd = ID/EX, Register Rt; 2a. MEM/WB. Register Rd = ID/EX, Register RS; 2b. MEM/WB. Register Rd = ID/EX, Register Rt				
O p 5	0 1 1 0 0 0 0 0	Forward A = 00 ID/EX ALU の第1オペランドがレジスタ・フィールドから得られる	1c. LW/XSTでも, 2a, 2b を使うことはある				
O p 4	0 0 0 0 0 0 0 0	Forward A = 10 EX/MEM ALU の第1オペランドが前のALUの結果が送られる					
O p 3	0 0 1 0 1 1 1 1	Forward B = 00 ID/EX ALU の第2オペランドがレジスタ・フィールドから得られる					
O p 2	0 0 0 1 0 1 1 0	Forward B = 10 EX/MEM ALU の第2オペランドが前のALUの結果が送られる					
O p 1	0 1 1 0 0 0 0 1	Forward B = 01 MEM/WB .. 2つ前の ..					
O p 0	0 1 1 0 0 0 1 0	Forward B = 01 MEM/WB .. 2つ前の ..					
RegDst	1 0 X X 0 0 0 0	shift + left logical; sll \$s1, \$s2, 10 s1 = s2 < 10 定数分左シフト					
ALUSrc	0 1 1 0 1 1 1 1	shift + right logical; srl \$s1, \$s2, 10 s1 = s2 > 10 定数分右シフト					
MemtoReg	0 1 X X 0 0 0 0	branch on equal; beq \$s1, \$s2, 25 if (\$s1 == \$s2) go to PC+4+100 等しいときにPC相対分歧					
RegWrite	1 1 0 0 1 1 1 1	branch on not eq; bne \$s1, \$s2, 25 if (\$s1 != \$s2) go to PC+4+100 等しくないときにPC相対分歧					
MemRead	0 1 0 0 0 0 0 0	set on less than slt \$s1, \$s2, 3 if (\$s2 < \$s3) \$s1 = 1; else \$s1 = 0 よりlt+tuが判定. beg & bne用					
MemWrite	0 0 1 0 0 0 0 0	set on less than immediate slti \$s1, \$s2, 20 if (\$s2 < 20) \$s1 = 1; else \$s1 = 0 定数より小さいかの判定. 2の補数					
Branch	0 0 0 1 0 0 0 0	OP: ALUAction = {lw: add, sw: add, beg: subtract, AND: and, OR: or, sset: set less than, le: set less than}					
ALUOp1	1 0 0 0 1 1 1 1	ID/EXでハザードを検出したならば, ID/EXパipelineレジスタ中の EX, MEM, WB (= 9つの制御線) の各制御フィールドを0に変更することによって, パソコン内にハザードを抑止できる。すなはち制御線がすべて0ならば、レジスタにモードが書き込みは行われない					
ALUOp0	0 0 0 1 1 1 1 1	平均アクセス時間 = $A \cdot MAT \div \text{クロック周波数}$					
MAT (平均メモリ・アクセス時間)(単位: CCP時間) = ヒットした場合のアクセス時間 + ミス率 × ミス・ペナルティ							
ミス・ペナルティ = キャッシュ時のストップクロックサイクル数 = 主記憶のアクセスタイム + クロック周期							
CCPの実効CP = 基本CP + [キャッシュのミス率(命令) × ミスペナルティ + キャッシュのミス率(データ) × アクセス比率 × ミスペナルティ]							
キャッシュボトルネcks (2^m+2 バイト)。したがって、ブロック内の語にmビット、アドレスのバイト部分に2ビットをそれぞれ使用する							
波動とは、波動や振動が単位時間あたりに繰り返される回数のこと。単位はHz。周期とは、一定時間毎に同じ現象が繰り返される場合のその一定時間のこと							

Hamming 誤り訂正コード(ECC) → 1. カラム右にビット番号を1から順に振る; 2. 0のべき乗であるすべての位置(1, 2, 4, 8, 16, ...)のビットをパリティビットにする; 3. それ以外のビット位置をデータ用に使用する; 4. パリティビットの位置によって、そのブロック対象の位置のデータの範囲が決まる → ビット1は、ビット(1, 3, 5, 7, 9, 11, ...)をチェックする; ビット2は、ビット(2, 3, 6, 7, 10, 11, 14, 15)をチェックする; ビット4は、ビット(4~7, 12~15, 20~23, ...)をチェックする; ビット8は、ビット(8~15, 24~31, 40~47, ...)をチェックする; 5: 各ブロックにおいて、何枚パリティとなるようにパリティビットを設定する
 ファイルマップチャート

$$\text{インデックス部ビット幅} = \lfloor \log_2 32 \text{bit} = 5 \text{byte} \rfloor; \text{1 block} = 2 \text{word}; \text{インデックス部ビット幅} = 5 (\text{キャッシュ容量} = 32 \text{block})$$

$$= 2^5 \text{block} \text{より}; \text{タグ部ビット幅} = 24 (\text{タグ部ビット幅} = 32 - (n+m); n = \text{インデックス部ビット幅} = 5)$$

1. ブロックサイズが $2 = 2^1 \text{字}(word)$ より $m = 1+2 = 3$; またにタグ部ビット幅 = $32 - (5+3) = 24$
 $32 \text{キャッシュ全体の総ビット数} = \text{ブロック数} \times (\text{ブロックサイズ} + \text{タグ長} + \text{有効アドレス長}) = 32 \times (2 \times 32 + 24 + 1)$
 $\text{block} = 32 \times 8 = 256$; Q. 次のアドレス範囲でアクセスされたとき、表を埋めよ ⇒ 1. ブロック番号を求める
 $\text{ブロック番号} = [\text{address}/\text{1ブロックの容量}(\text{byte})] \text{の整数部分}$ 2. Index を求める: $\text{Index} = [\text{block番号}/\text{ブロック数}] \text{の余り}$
 3. Tag を求める: $\text{Tag} = [\text{block番号}/\text{ブロック数}] \text{の整数部分 ex. address block番号 Index Tag H/M}$
 $\text{tag}([\text{Index}]) = [\text{Tag}]$

セレクトアンドアディショナルキャッシュ

ブロック16byte, 8set, 3way; インデックス部ビット幅 = 3 (set数 = 8 = 2^3 より 3bit); タグ部ビット幅 = 25 (タグ部ビット幅 = 32 - (n+m); n = インデックス部ビット幅 = 3; 1block = 16byte = 2^4 より, m = 4; クエリタグ部ビット幅 = 32 - (3+4))

ブロック番号 = $(\text{address}/\text{1ブロックの容量}(\text{byte}))$ の整数部分; 2. Index を求める: $\text{Index} = [\text{block番号}/\text{ブロック数}] \text{の余り}$
 整数部分; 3. Tag を求める: $\text{Tag} = [\text{block番号}/\text{ビット数}] \text{の整数部分}$

タグ([Index]) = [Tag]

※ ヒットした場合は、他の書き込みは発生しない

for (i = 0; i != 10; i++) {
 sum += data[i];
 }

但し, i = \$t0, data = \$s0, sum = \$s1 とする
 100 add \$t0, \$zero, \$zero # i = 0とする
 104 addi \$t1, \$zero, 10 # i = 10が範囲外となる
 108 beg \$t0, \$t1, ? # i = 10が範囲外となる
 10c sll \$t2, \$t0, 2
 110 add \$t2, \$t2, \$s0
 114 lw \$t3, 0(\$t2)
 118 add \$s1, \$s1, \$t3
 11c addi \$t0, \$t0, 1
 120 beg \$zero, \$zero, -7 # 108に戻る

```

if (data[i] < data[i+1])
    data[i] = 0;
else
    data[i] = data[i+1];
return (i);

```

但し i = \$t0, data = \$s0, 10進数で表示すると \$t0 = 0, \$s0 = 104

```

1. sll $t1, $t0, 2
2. add $t1, $t1, $s0
3. lw $t2, 0($t1)
4. lw $t3, 4($t1)
5. sll $t4, $t2, 3
6. beg $t4, $zero, 2
7. sw $zero, 0($t1)
8. beg $zero, $zero, 1
9. sw $t3, 0($t2)
10. addi $t0, $t0, 0

```

ten 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
hex 0 1 2 3 4 5 6 7 8 9 A B C D E F
bin 0 1 10 11 100 101 110 111 1000 1001 1010 1011 1100 1101 1110 1111

ten 16 ※ 2進数が16進数で表示する(10進数で表示しない)の
hex 10 小書き

bin 10000 ライト・スルー方式 … データ書き込み際にトランジスタ記憶装置の次の下位レベルの両方を常に更新する方式。両者の間でデータが一貫していざといが常に保護される/ライト・バンク方式 … データ書き込み際はキャッシュ中のブロックを更新しないだけで、ブロックを置き換えるときに更新された内容を記憶装置の下位レベルに書き戻す方式

MUX(マルチセレクタ) … 2つ以上の入力を1つの信号として出力する機構。

