MIPSの性能評価

慶應義塾大学
天野

シングルサイクル版vs.
マルチサイクル版

- CPUのマイクロアーキテクチャは
  性能、コスト（面積）、消費電力で評価する。
- ここでは性能とコスト（ハードウェア量）を簡単に評価する。
- 本格的な評価には論理合成が必要→来年
- 本当はこれに電力も加わるが、これも来年

では、次にシングルサイクル版とマルチサイクル版のどちらのマイクロアーキテクチャが有利なのかを評価しましょう。ここでは性能とハードウェア量を簡単に見積もって比較しましょう。本格的な評価は論理合成をやった後で、多分来年のコンピュータアーキテクチャになります。

CPUの性能評価式

- CPUの性能はプログラム実行時間の逆数

CPU Time＝プログラム実行時のサイクル数×クロック周期
＝命令数×平均CPI×クロック周期

CPI （Clock cycles Per Instruction） 命令当たりのクロック数

→ 1サイクル版では1だがマルチサイクル版では命令によって違ってくる

命令数は実行するプログラム、コンパイラ、命令セットに依存

では、次に性能の評価についての一般的な方法を学びます。CPUの性能は、CPUがあるプログラムを実行した際の実行時間の逆数です。実行時間が短い方が性能が高いのでこれは当たり前かと思います。実際のコンピュータではOperating System（OS)が走って実行中にもジョブが切り替わりますが、この影響が入ると困るので、CPUが単一のジョブをOSの介入なしに実行した場合の実行時間（CPU実行時間：CPUTime)を測ります。今まで紹介してきたように、CPUは単一のシステムクロックに同期して動くと考えて良いので、CPU Timeはプログラム実行時のサイクル数×クロック周期で表されます。クロック周期とはクロックが立ち上がってから次に立ち上がるまでの時間で、この逆数がクロック周波数です。プログラム実行時のサイクル数は、実行した命令数×平均CPI（Clock cycles Per Instruction)に分解されます。CPIは一命令が実行するのに要するクロック数で、mipse1サイクル版では全部1ですが、マルチサイクル版では命令毎に違っています。このため、一つのプログラムを動かした場合の平均CPIは、プログラムの種類によって変わります。つまり実行時間の長い命令を多数含んでいるプログラムでは平均CPIは長くなります。もちろんコンパイラにも依存します。

性能の比較

- CPU　A　10秒で実行
- CPU　B　12秒で実行
- Aの性能はBの性能の1.2倍
  遅い方の性能（速い方の実行時間）を基準にする

$$\frac{\text{CPU　Aの性能}}{\text{CPU　Bの性能}} = \frac{\text{CPU　Bの実行時間}}{\text{CPU　Aの実行時間}}$$

×BはAの1.2倍遅い　この言い方は避ける

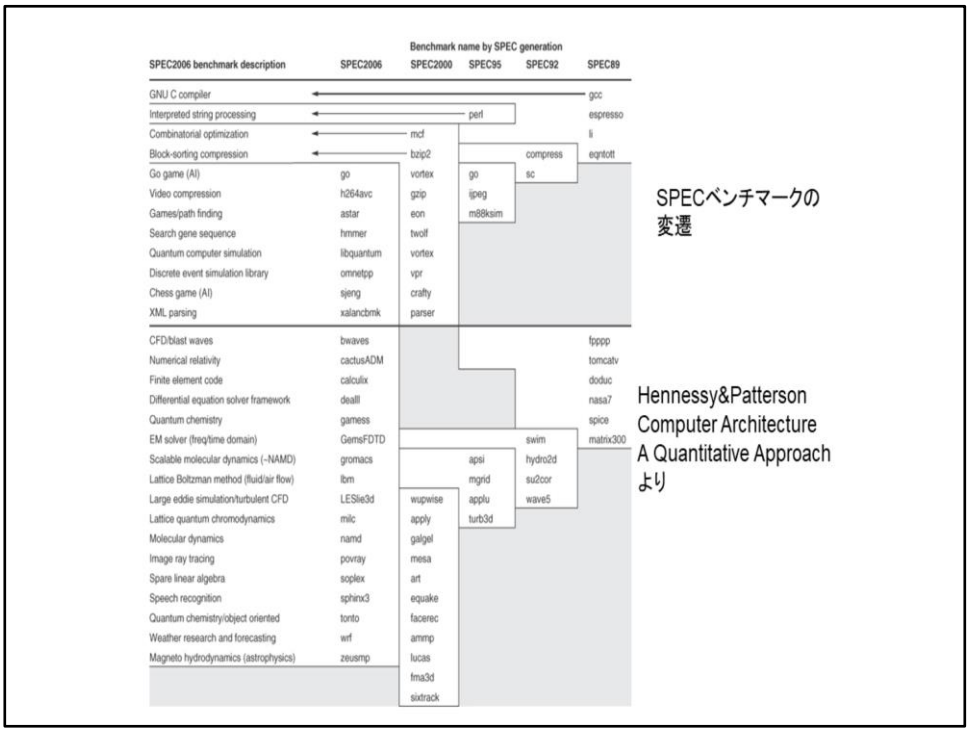
では次に性能の比較方法について検討します。CPU Aはあるプログラムを10秒で実行し、Bは同じプログラムを12秒で実行します。AはBの何倍速いでしょう？この場合、Bの性能を基準とします。Bの性能はBの実行時間の逆数、Aの性能はAの実行時間の逆数なんで分子と分母が入れ替わり、Bの実行時間をAの実行時間で割った値となります。これは12／10で1.2倍になります。ではBはAの何倍遅いのでしょうか？この考え方は基準が入れ替わってしまうため混乱を招きます。このため、コンピュータの性能比較では常に遅い方の性能（つまり速い方の実行時間）を基準に取ってで、（速い方）は（遅い方）のX倍という言い方をします。

実行時間の評価

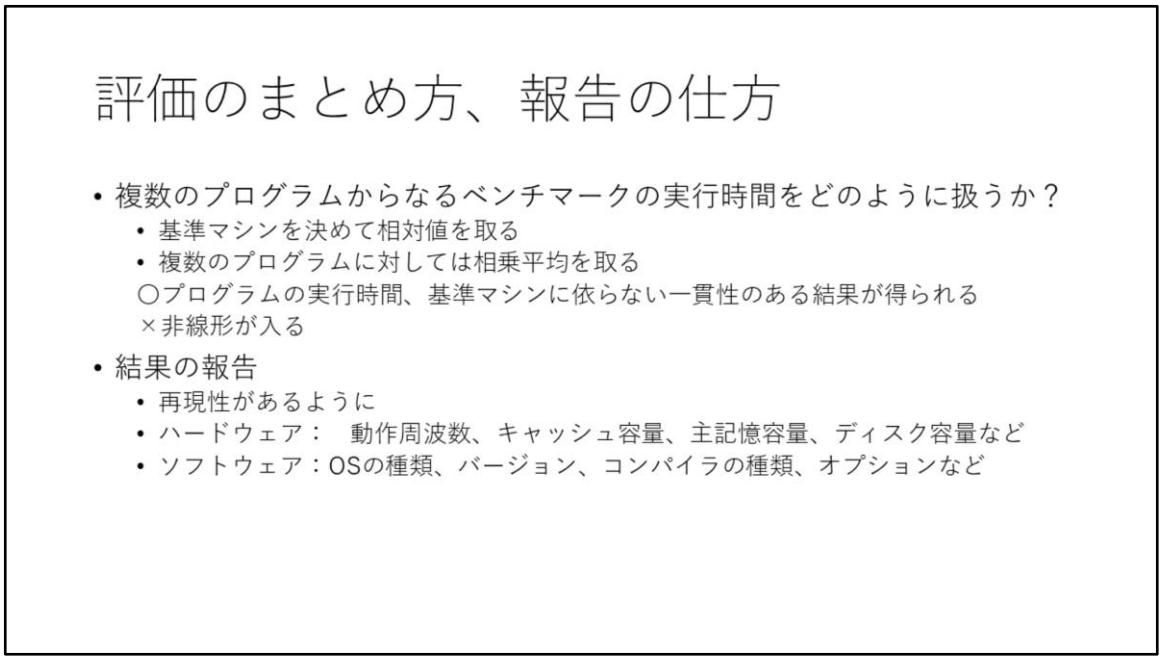
- プログラムを走らせてその実行時間を比較
  - デスクトップ、ラップトップ：ＳＰＥＣベンチマーク
  - サーバー：TPC
  - スーパーコンピュータ：Linpack, LLL
  - 組み込み：EEMBC, MiBench
- 走らせるプログラム
  - ○実プログラムによるベンチマーク集
  - △カーネル：プログラムの核となる部分
  - ×トイプログラム：Quicksort, 8queen, エラトステネスの篩
  - ×合成ベンチマーク：Whetstone, Dhrystone

では、実行時間はどのように評価すればよいでしょうか？もちろんプログラムを走らせればいいのですが、ではどのようなプログラムを走らせればいいのでしょう？最も良く使われているのは、現実のアプリケーションプログラムを一定の入力データのセットと組み合わせたベンチマーク集です。ベンチマーク集（ベンチマークスーツ）は複数のプログラムからできていて、良く使われるのがデスクトップやラップトップの業界で使われるSPECベンチマークです。SPECベンチマークはCコンパイラ、CAD、人工知能のプログラムから成る非数値系のSPECintと、流体力学、構造解析、量子力学などの数値計算のプログラムから成るSPECfpがあります。実プログラムから出来ているため、リアルな評価ができる利点がありますが、評価するのに手間が掛かり、プログラムが複雑なので具体的な性能の分析がやり難いです。このため、スーパーコンピュータや組み込みシステムでは、実際のプログラムの中で良く使われる部分を抜き出したカーネル（プログラム核）を用いる場合があります。スーパーコンピュータのランキングに使うLinpackなどのがこの例です。Quicksort、８－Queenなどの簡単なプログラム（トイプログラム）は、コンパイラがまだ出来ていない計算機の評価に使われる場合もありますが、一般的なプログラムとは違った特殊な動きをするため、あまり良い方法ではないです。また、様々なプログラムの挙動を一つに詰め込んだ合成ベンチマーク、Whetstone,Dhrystoneは、今でも組み込み分野では使われることがあります

が、やはりプログラムの挙動が一般的ではない点、ベンチマークのみに通じる最適化を施しやすい点などから、やはり良い方法ではないといわれています。



このスライドはSPECベンチマークの変遷を示しています。真ん中の線より上がSPECint、下がSPECfpです。ベンチマークは、対象とするマシンの性能向上やメモリの増大に対応して、一定の期間毎に入れ替えが行われています。



ではベンチマーク集を使って評価をしたとしましょう。複数のプログラムからなるベンチマークの実行時間をどのようにまとめればよいのでしょう？それぞれのベンチマークの実行時間を同じにすることはできません。単純に実行時間の平均を取ると、実行時間の長いプログラムのウェイトが大きいことになってしまいます。しかし、ベンチマークの実行時間は入力データとの関係で決まり、それが長いからと言って全体に対する影響力が大きいとはいえません。そこで、まず、皆が持っているマシンを基準マシンとし、評価するマシンとの相対値を取ります。多数のプログラムを実行した場合、この相対値の相乗平均（幾何平均）を取ります。この方法は、プログラムの実行時間が違っても、基準マシンが変わっても、皆が同じものを使えば、一貫性のある結果がでます。ただし、これで得られた結果は、あくまで目安に過ぎません。相乗平均には非線形性があるので、ベンチマークのプログラムを組み合わせて走らせた場合に平均的にこの数値分速くなることはないのです。
次に結果の報告については、他の人が同じマシンを使って同じプログラムを走らせた場合、同じ結果が出るように、つまり再現性があるように、ハードウェアの諸元をはじめ、ソフトウェアについてもOSの種類、バージョン、コンパイラの種類、オプションなどを報告するように心がけてください。

Description	Name	Instruction Count × 10⁹	CPI	Clock cycle time (seconds × 10⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPEC Ratio
Interpreted string processing	perl	2,118	0.75	0.4	637	9,770	15.3
Block-sorting compression	bzip2	2,389	0.85	0.4	817	9,650	11.8
GNU C compiler	gcc	1,050	1.72	0.4	724	8,050	11.1
Combinatorial optimization	mcf	336	10.00	0.4	1,345	9,120	6.8
Go game (AI)	go	1,658	1.09	0.4	721	10,490	14.6
Search gene sequence	hmmer	2,783	0.80	0.4	890	9,330	10.5
Chess game (AI)	sjeng	2,176	0.96	0.4	837	12,100	14.5
Quantum computer simulation	libquantum	1,623	1.61	0.4	1,047	20,720	19.8
Video compression	h264avc	3,102	0.80	0.4	993	22,130	22.3
Discrete event simulation library	omnetpp	587	2.94	0.4	690	6,250	9.1
Games/path finding	astar	1,082	1.79	0.4	773	7,020	9.1
XML parsing	xalancbmk	1,058	2.70	0.4	1,143	6,900	6.0
Geometric Mean							11.7

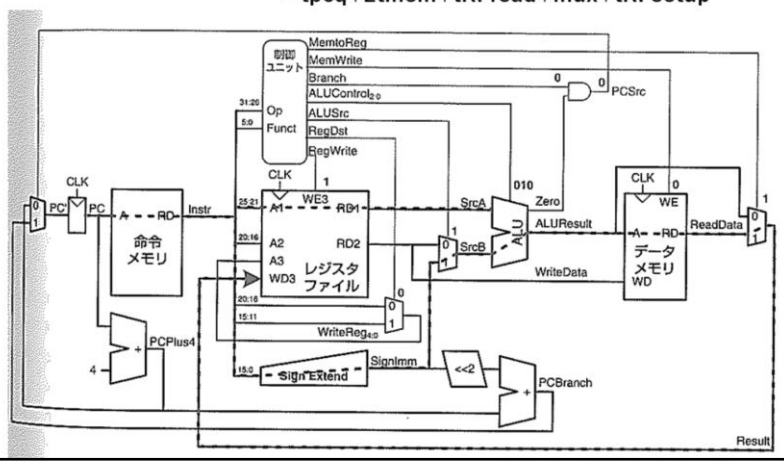
FIGURE 1.20 SPECINTC2006 benchmarks running on AMD Opteron X4 model 2356 (Barcelona). As the equation on page 35 explains, execution time is the product of the three factors in this table: instruction count in billions, clocks per instruction (CPI), and clock cycle time in nanoseconds. SPECratio is simply the reference time, which is supplied by SPEC, divided by the measured execution time. The single number quoted as SPECINTC2006 is the geometric mean of the SPECratios. Figure 5.40 on page 542 shows that mcf, libquantum, omnetpp, and xalancbmk have relatively high CPIs because they have high cache miss rates.

この表はSPECINT2006ベンチマークをOpteran X4モデル2356というマシンで走らせた場合の性能を示しています。ここで、SPECratio（一番右の欄）が基準マシンとの性能（実行時間の逆数）の比ですがプログラムの種類によって非常に違っていることがわかります。最後にGeometric Mean（相乗平均）が示されています。

シングルサイクル版のクリティカルパス

- lw命令が最も長い

tpcq+tmem+tRFread+tALU+tmem+tmux+tRFsetup
=tpcq+2tmem+tRFread+mux+tRFsetup



では、このCPUの性能を見積もってみましょう。シングルサイクルマイクロアーキテクチャは、全ての命令を1クロックサイクルで終わらせるので、最も遅延時間の長い命令の遅延を調べれば動作周波数が分かります。CPI=1なので、性能は動作周波数で決まります。ではどの命令の遅延パスが一番長いでしょうか？それはALUで実効アドレスを計算して、これでデータメモリを読み出すlw命令です。最も長い遅延パスをクリティカルパスと呼びます。これは図に示すようになります。

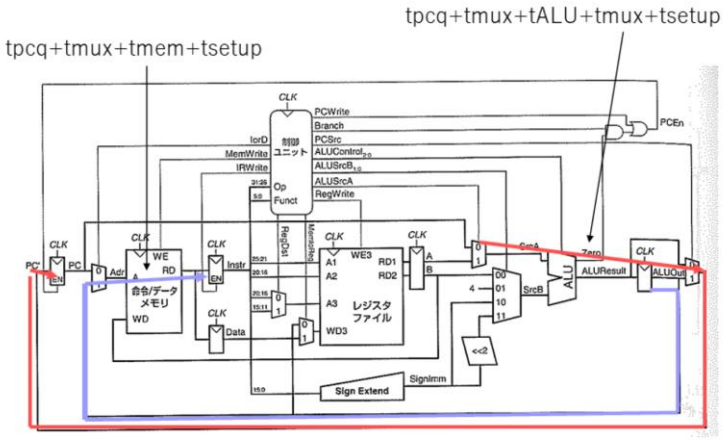
遅延の例

遅延要因	記号	遅延(psec)
レジスタclk→Q	tpcq	30
レジスタセットアップ	tsetup	20
マルチプレクサ	tmux	25
ALU	tALU	200
メモリ読み出し	tmem	250
レジスタファイル読み出し	tRFread	150
レジスタファイルセットアップ	tRFsetup	20

この数値を使うと30+2(250)+150+25+200+20=925psec
1.08GHzとなる。

この表は各部の遅延時間の例です。遅延時間はCPUを実装するプロセスによって決まりますが、この値は最近のプロセスとしてリーゾナブルなものです。やはり、メモリの読み出し時間が長いです。ALUは演算機の作り方によりますが、これに次ぐ長さになります。この数値を使うとクリティカルパスは925psecとなり、1.08GH z で動作することがわかります。

マルチサイクル版の
クリティカルパスの検討



マルチサイクル版のマイクロアーキテクチャでも、最も長いパスがクリティカルパスになります。これはシングルサイクル版よりも短くなります。データパスを検討すると、この2つのパスの辺になりそうです。

動作周波数（周期）の解析

- クリティカルパス：今回の仮定では
  - ALU：tpcq+tmux+tALU+tmux+tsetup
    30+25+200+25+20＝300ps
  - メモリ：tpcq+tmux+tmem+tsetup
    30+25+250+20=325ps
- メモリの遅延が大きくなる：325ps→3.07GHz
- これは、シングルサイクルの1.08GHzよりもかなり高くなっている。

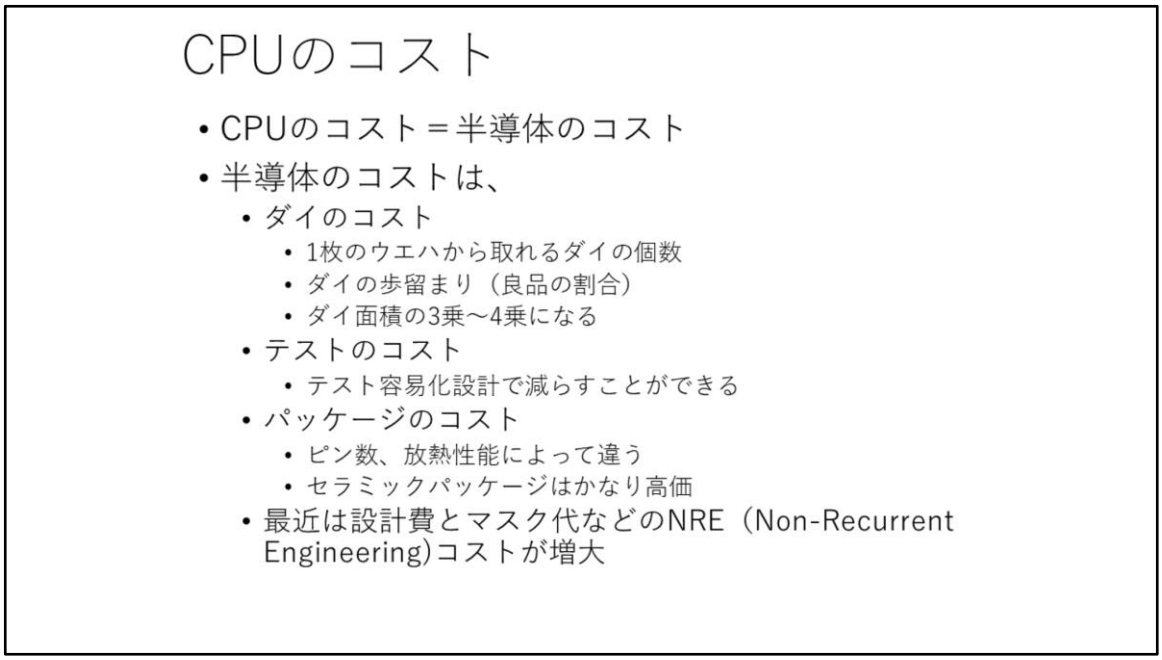
→クリティカルパスを切断した効果

この二つのパスを先ほどの値を入れて検討するとこのようになります。今回はメモリの遅延が大きいため、325psが動作周期となり、したがって動作周波数はこの逆数を取って、3.077GH z となります。この数値は、シングルサイクルよりもかなり高くなっており、クリティカルパスを切断した効果が表れています。

CPIを測ろう

• mult.asmをアセンブルして、実行クロック数と実行命令数を測る。
make mult　　multのアセンブル
make　　iverilog
./mipse
でプログラムが停止するとClock Countに実行クロック数、Inst Countに実行命令数が表示される。
CPI＝実行クロック数/実行命令数
プログラムは0x2000番地に何か書くと停止するようにできている。
実行時間＝周期×CPI
で、実行時間を求めよう。
シングルサイクル版のCPI＝１、実行命令数は同じである。この
実行時間を求め、比較してみよう。

では、実際のプログラムを実行してみましょう。手順に従って実行すると、CPIを求めることができます。さらに、先に求めた周期から実行時間が分かります。シングルサイクル版のCPIは１で、実行命令数は同じです。この実行時間を求めて比較してみましょう。



CPUのコストは、半導体のコストによって決まります。半導体のコストは、ダイ（次のページの図）のコスト＋テストのコスト＋パッケージのコストで計算できます。ダイのコストは、ウエーハ1枚のコストを（1枚のウエーハ（次のページの図）から取れるダイの個数とダイの歩留まり（良品の割合）の積）で割ったものになります。ダイの面積が増えるほど、1枚当たりから取れる数が減ります。また、ダイの歩留まりは、半導体の欠損がどの程度発生するかによって決まるのですが、やはり面積が大きくなるほど悪くなります。ざっくり考えてダイのコストはダイの面積の3乗から4乗になると言われています。しかし、一部に欠損があっても動作するように設計する冗長設計によって、歩留まりは改善することができます。半導体は高額なテスターを使って、正しく動作するかチェックします。このコストも馬鹿にならない位大きくなります。これはテスト容易化設計で、テスト工程を簡単にすることで減らすことができます。さらにパッケージのコストも掛かります。これは、ピン数の多く放熱特性に優れたセラミックパッケージを使う場合、高額になります。電力とピン数を削減してプラスチックの安いパッケージにすることができれば削減ができます。最近の新しい半導体プロセス技術を使うと、設計費、IP代、マスク代などのNRE（Non-Recurrent Engineering)コスト、すなわち一回だけ掛かる製造費が非常に大きくなっています。

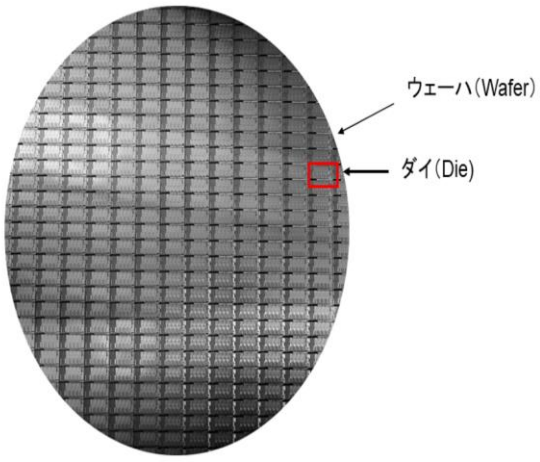
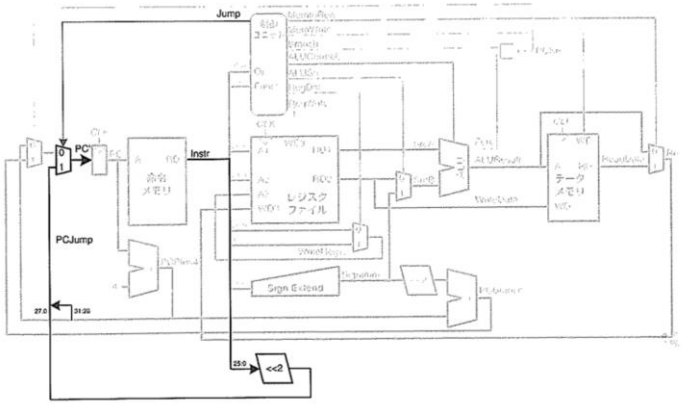


Figure 1.15 This 300 mm wafer contains 280 full Sandy Bridge dies, each 20.7 by 10.5 mm in a 32 nm process. (Sandy Bridge is Intel's successor to Nehalem used in the Core i7.) At 216 mm2, the formula for dies per wafer estimates 282. (Courtesy Intel.)

この図はIntelのCore i7(Sandy Bridge)のウェーハ写真です。直径30センチの円盤上に長方形のダイが並んでいます。これを切り離して、パッケージに組み込んで半導体チップができます。周辺部の模様が欠けているダイはもちろん使えません。ウェーハは半導体の製造工程上、どうしても30センチ程度の円盤になるので、ダイの面積が増えると、搭載できる個数が減ってしまうことがわかります。

コストの計算：シングルサイクル版

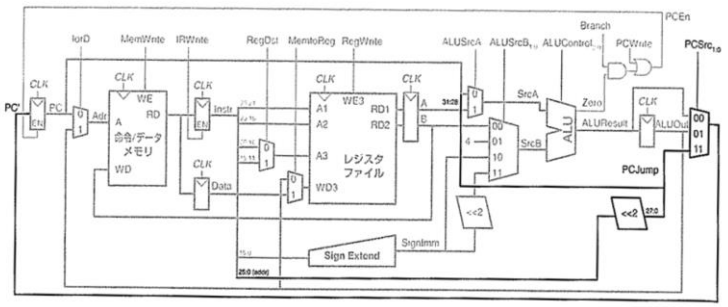
モジュール	個数
メモリ	2
レジスタファイル	1
ALU	1
加算器	2
マルチプレクサ	5
レジスタ	1



今回は半導体のコストまでは見積もれないので、その前段階となるモジュール数を見積もって見ましょう。シングルサイクル版ではj命令を実装した段階でのデータパスのリソース使用量は表のようになっています。

コストの計算：マルチサイクル版

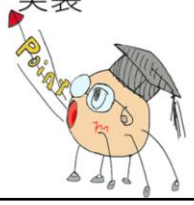
モジュール	個数
メモリ	1
レジスタファイル	1
ALU	1
加算器	0
マルチプレクサ	6
レジスタ	6



一方、マルチサイクル版は、メモリが一つで済み、加算器がなくなった一方、マルチプレクサが増え（入力数も増えています）、レジスタも増えています。とはいえ、マルチプレクサ、レジスタのハードウェア量はさほど大きくないことを考えると、コスト的にはかなり有利と言えると思います。ただし、このコストにはFSMのは含まれていないので注意は必要です。

性能とコストの比較のまとめ

- ISAが同じ場合、性能は、クロック周期とCPIで決まる。
  - クロック周期はクリティカルパスで決まる。
  - CPI（Clock cycles Per Instruction)は、シングルサイクル版は常に１だがマルチサイクル版は動作させるプログラムに依存
  - 性能比較は、遅い方の性能（速い方の実行時間）を基準にする。
- コストは必要モジュール数で評価したが、実装の状況により異なる。



性能とコストの比較の部分をまとめます。

演習1　性能評価

- 0x1000番地から並んでいる8個のデータの総和を求めるプログラムmsum.asmを実行し、マルチサイクル版のCPIを求めよ
- この値と授業中のスライドの数値を利用して、シングルサイクル版mipseとマルチサイクル版のmipseの性能を比較せよ。

XXがYYのZZ倍速い、という言い方で示せ。

最初は楽勝です。

演習２　相乗平均による比較

- 授業中の演習で求めたmultと演習1で求めたmsumの相乗平均を求めて、シングルサイクル版と、マルチサイクル版の平均的な速度比を計算せよ。

→これは単なる計算問題