[강좌] 가정부로 이해한다. CPU 구조 -2부.

가정부로 이해한다. CPU 구조 -2부.

CPU 와 메인 메모리, 레지스터, 메인 메모리의 중요한 관계

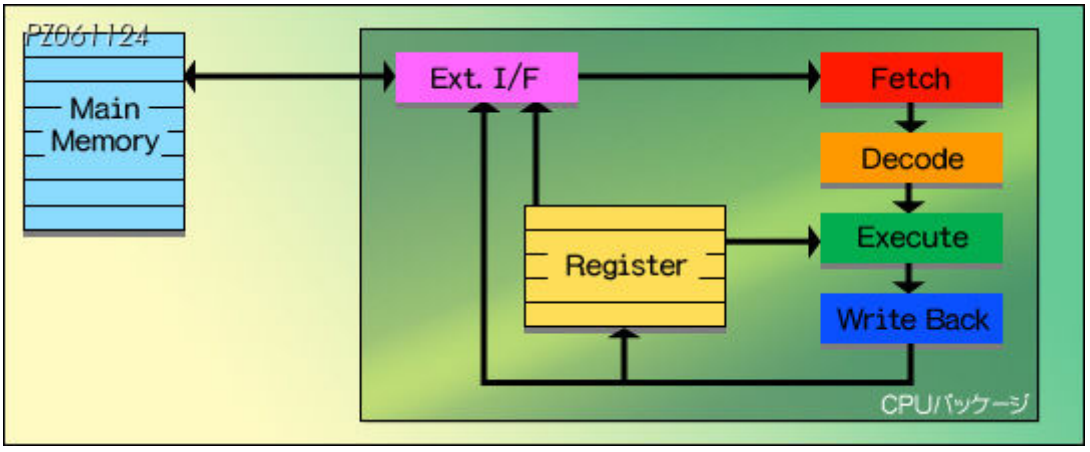
앞선 1 회에서 설명한 내용은 CPU의 구조와 그것을 얼마나 고속화하는가 하는 것이었지만, CPU 그것"만"빠르게 동작해도 고속화에는 한계가 있다. 이전처럼 가정부가 등장하여, 예를 들면 저택의 주인으로부터「XX이라고 하는 와인을 가져오」라고 하는 명령이 나왔을 때, 그 와인이 저택에서 멀고 먼 항구의 창고에 있었다고 하자. 이렇게 되면, 가정부가 만일 통상의3 배의 속도로 움직일 수 있었다고 해도, 교통 수단이 1 배 속도라면, 항에 가서 돌아올 만한 시간은 그대로 주인에게 있어서 대기 시간이 된다.이동중에 다른 작업을 할 수 있다면, 가정부의 전체적인 일 시간은 단축할 수 있지만, 세상 그렇게 잘 된다고는 할 수 없다.이 문제를 발본적으로 해결하기 위해서는, 와인을 배달시켜 오는 과정도 3배의 속도로 처리해야 하는 것이다.

그럼, CPU에 있어서의 이런 문제에 대한 해결책은「메인 메모리」(Main Memory),「레지스터」(Register),「캐쉬」(Cache) 라고 하는 기억장치가 이것에 상응한다.

메인 메모리는 잘 알려져 있는 존재일 것이다. PC 의 스펙으로PC3200 (DDR400)(이)라든지PC2-4200 (DDR2 533)(이)라든지, 혹은512MB 라든가1GB 라든가 하는 예. 또한 레지스터는 한마디로 하면, CPU 내부에 있어 일시적으로 데이터를 저장하는 기억장치로, 아래와 같은 특징이 있다.

고속 : CPU 코어와 같은 스피드로 동작해, 읽고 쓰기하는데 대기 시간은 전혀 없다.
소용량 : 예를 들면 x86의 경우, 범용적으로 이용할 수 있는 것은 최대 데이터 8개만.
데이터 전용 : 프로그램의 저장은 할 수 없고, 데이터만 저장할 수 있다.

초기의 CPU에 있어서의, CPU 코어(1장에서 설명한, 페치로부터 write back까지의 흐름을 담당하는 부분이라고 생각하면 된다) 와 메인 메모리, 레지스터의 관계는 그림1처럼 무엇인가 처리를 실시할 때, CPU 는 외부 인터페이스(그림1의 Ext.I/F) 경유로 메인 메모리로부터 프로그램을 읽어낸다. 레지스터는 어디까지나 일시적인 기억 에리어이므로, 레지스터를 사용해 행해진 처리의 최종적인 결과는, 다시 메인 메모리에 써낼 필요가 있다.



(그림1)

이 단순한 구성은 1995년에 등장한「Intel 80386 」에서 시작 하였고, 이 CPU들은 문제가 없었다. 당시의CPU의 동작 클럭(=동작 주파수)은 최고 33MHz 정도인 한편, 당시의 메모리는 액세스 속도가 100ns (10MHz)~80ns (12.5MHz)에 이르고 있어CPU 의 동작 클럭 보다는 다소 늦기는 했지만, 그만큼 큰 갭은 아니었기 때

문이다.

문제가 되는 것은 이 다음에 있다. 파이프라인화를 시작으로 하는 고속화 기법이나, 프로세스의 미세화등의 효과로 CPU의 스피드가 자꾸자꾸 올라가게 되어 메인 메모리의 속도가 거기에 따라잡아 갈 수 없는 상황을 맞이했던 것이다. 당초 메인 메모리는 FPDRAM(Fast Page DRAM)이라고 하는 방식이 채용되고 있고, 최종적으로60ns (16.67MHz) 정도까지 고속화 되었지만, 거의 사용이 되지 않았다. 거기서 SDRAM (Syncronous DRAM)이라고 하는 새로운 구조가 채용되어 동작 클럭도 66MHz ~100MHz 정도까지 끌어 올려졌지만, 이 시점에서CPU 의 동작 클럭은 수백MHz 까지 오르고 있었다. SDRAM은 최종적으로 166MHz 정도까지 달했지만, 때를 거의 같이 한 CPU는 동작 클럭을 1GHz 추월하여 상황은 변함이 없었다. 게다가DDR SDRAM (Double Data Rate SDRAM)이든지 DDR2 SDRAM이든지와 신기술은 차례차례로 투입되고 있지만, 상황 개선의 전망은2005 년 가을 현재에도 전혀 나아지지 않고 있다

이것을 커버할 수 있도록 ,Intel 80386의 후계CPU인「Intel 80486 」(와 일부의Intel 80386 호환 제품) 부터 채용이 시작된 것이 캐쉬다. 캐쉬는 레지스터보다 좀 더 큰 용량(예를 들면Intel 80486 의 경우,8KB)를 가지는 일시 기억 영역에서, 프로그램과 데이터의 모두 저장할 수 있다. 캐쉬에 따른 효과는 그림3 ,4 에 정리했지만, CPU와 검은 선으로 연결되어 있는 사각이 메인 메모리라고 생각하면 좋겠다. 메인 메모리는 어쨌든 늦기 때문에, 메인 메모리내에 있는(1)~(4)의 데이터(그림중에서는「○」으로 표시)을 순서에 읽어내려고 하면, 그림3의 오른쪽으로 도표로 나타내 보인 것처럼, 하나 읽어낼 때 마다"메모리 대기"가 생겨 버린다. 그야말로, 첫머리에서 예시한 항구의 창고에, 가정부를 사용하러 하고 있는 것하고 독같은 상황이다.

그래서, 항구의 창고를 저택에 있는 빈 방으로 옮겨 버리면 된다. 이와 같이, 메인 메모리보다 CPU 에 가까운 장소에 놓여지는 기억 영역이 캐쉬이다. 짐두는 용도의 방에는 어느 정도 결정된 양의 짐을 둘 수 있기 때문에,「항구의 창고에 있다. (1)를 가져오고」라고 하는 명령이 있었을 때에, (2)~(4)과정이 이행되고, 다음에「(2)를 가져오고」라고 하는 명령이 내려졌을 때에 곧 대응할 수 있다는 것이다(그림 4).

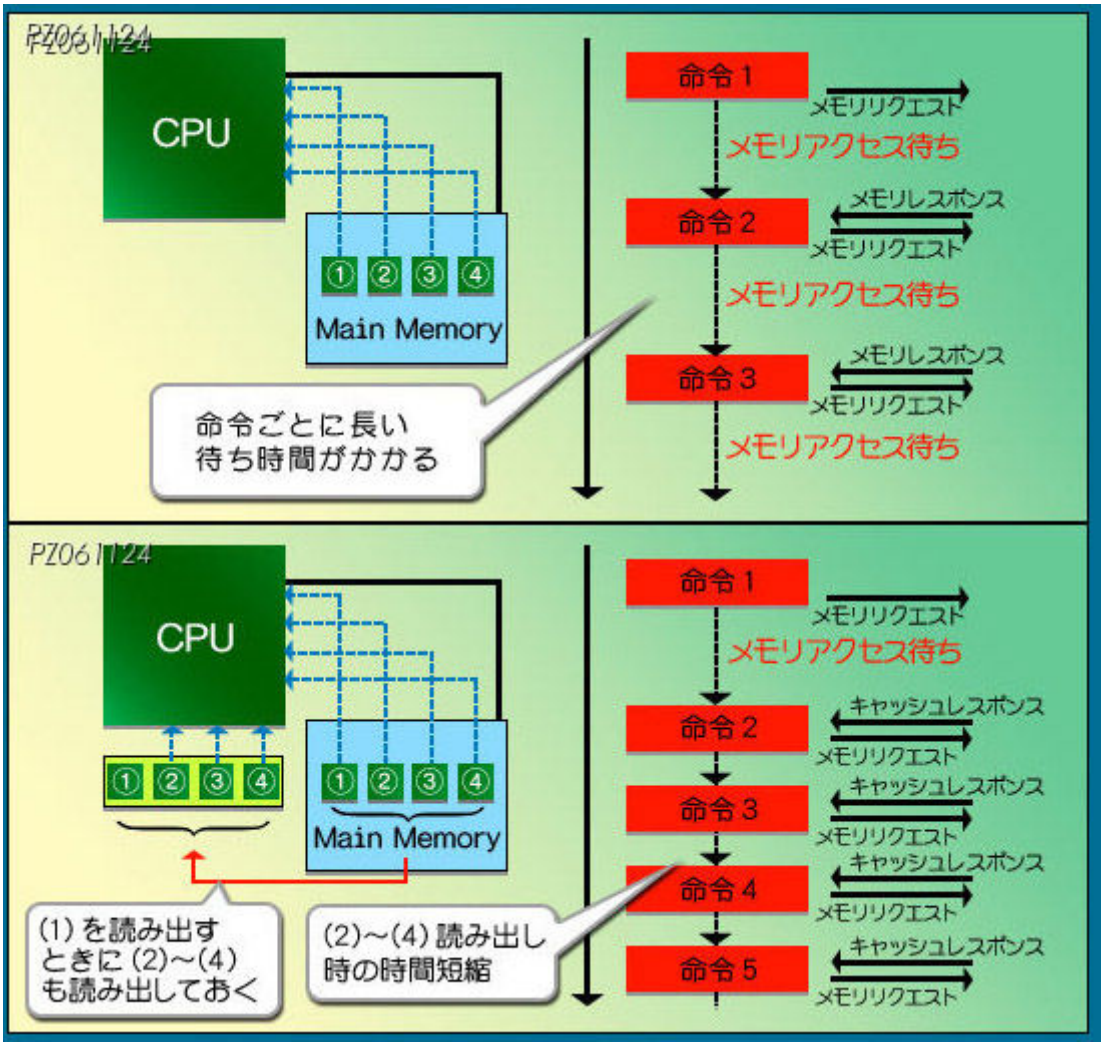
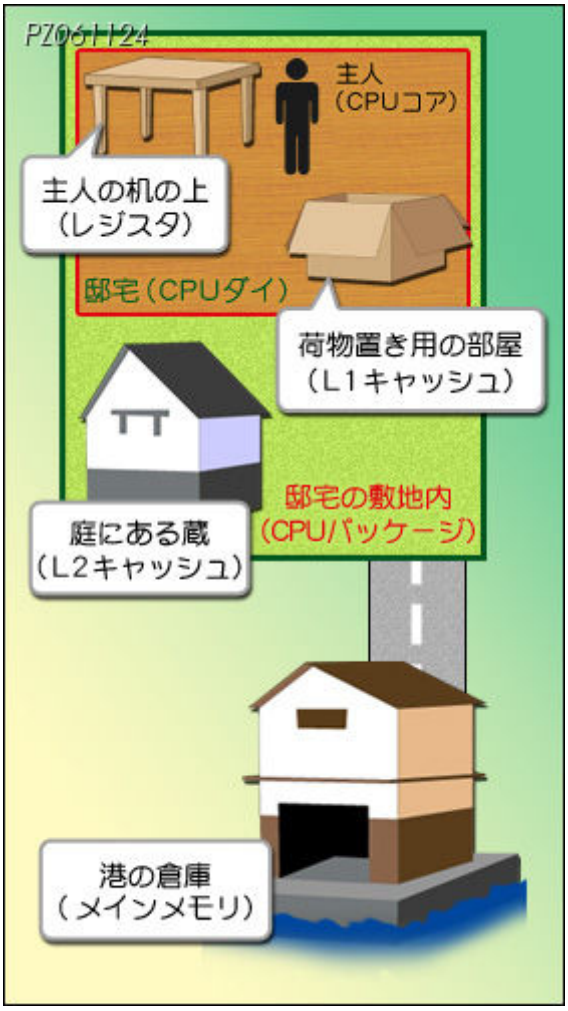


그림3,4

주인으로부터 가까운 곳에 짐두는 곳을 만드는 편이 대기 시간이 적게 되지만, 자꾸자꾸 창고로부터 짐두는 용도의 방에 옮겨 들이고 있으면, 당연히 방은 펑크나 버린다. 이러면 어떻게 하는가 하면, 저택의 뜰에 창고를 세우면 된다. 확실히 방으로부터 잡아 오는 것보다는 시간이 걸리지만, 창고까지 잡으러 가는 것보다는 단연 빠르기 때문이다.

CPU의 이야기에 되돌리면, CPU의 내부에 「L1 캐쉬」(Level 1 Cache)밖에,「L2 캐쉬」(Level 2 Cache)가 놓여지는 방식이다. 데이터나 명령을 수중에 넣을 필요가 있을 때, CPU는 우선 L1 캐쉬에 액세스 하고 거기에 없으면, L2 캐쉬에 액세스 한다. 거기에도 없을 때는 다시 메인 메모리에 액세스 하는 것이다(그림 5). 덧붙여서, CPU의 역사상에는, L3 캐쉬를 채용하는 제품도 있었다.



(그림 5)

위의 예로에서 나온「주인의 책상 위」정도가 레지스터와 비교된다. 그리고 L1 캐쉬는 CPU 코어와 같은 속도로 움직이지만, 레지스터와 비교해서 지연시간(읽기를 한 후, 실제로 받아 들여질 때 까지의 시간)이 여분으로 걸린다. 또L2 캐쉬(나L3 캐쉬)는, CPU 보다 훨씬 낮은 속도로 움직이게 되며, 실제로 수백MHz 또는 Pentium III나 Athlon시대는 CPU의 2분의 1, 3분의 1의 속도로 움직이는 것도 있었다. 또 지연시간도 L1 캐쉬 보다 더 커지는 것이 일반적이다. 하지만 그래도, 메인 메모리에 직접 액세스 하는 것보다는 수배~수백배 고속를 가지고 있다.

캐쉬 용량이 증가하지 않는 이유

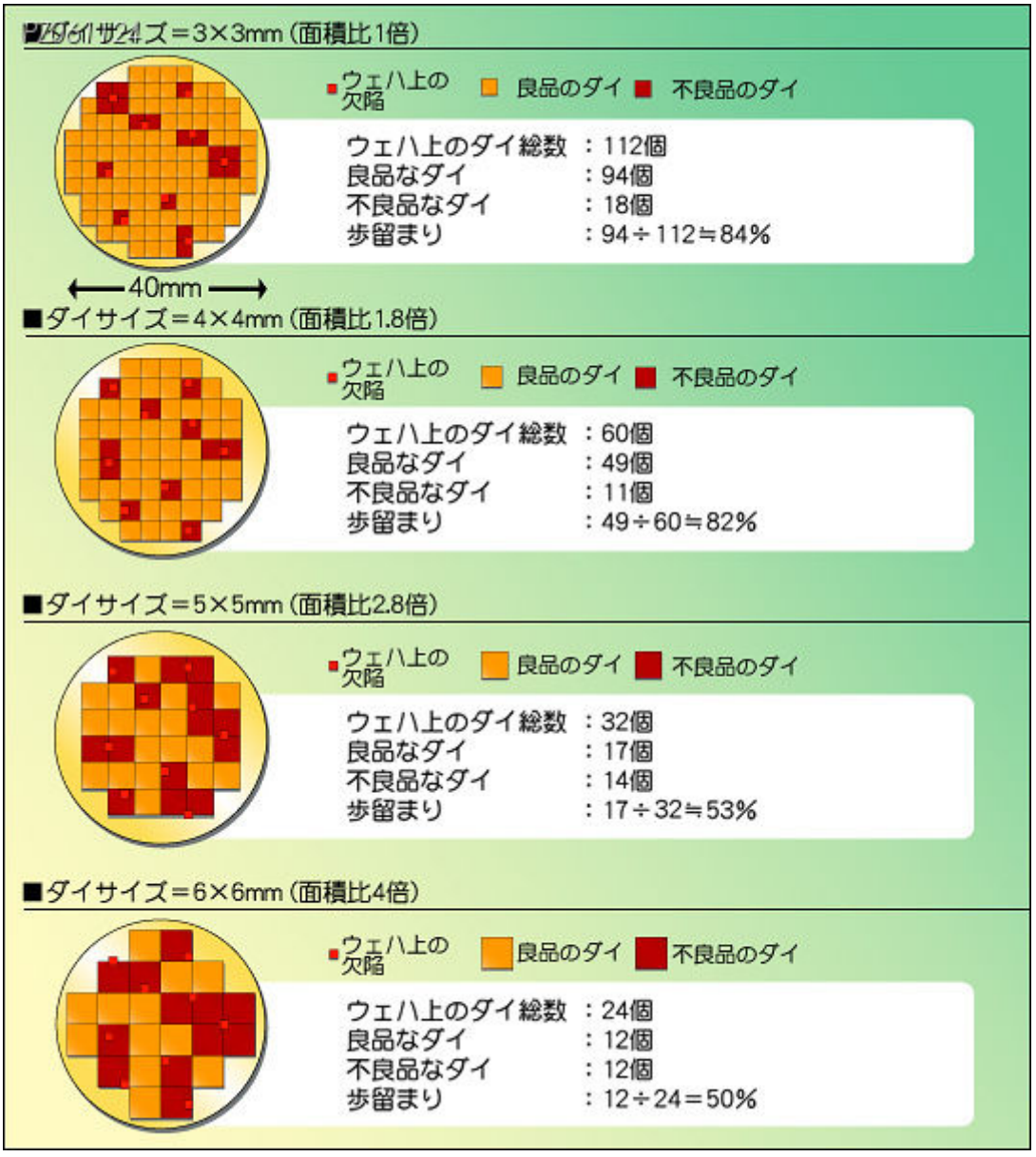
캐쉬가 그 만큼까지 성능이 좋다면, 메인 메모리등을 사용하지 않고 GB되는 캐쉬를 CPU 안에 두면 좋다고 생각하는 사람도 있을 것이다. 이것은 정도로서는 완전히 올바르다. 어쨌든 메인 메모리에 직접 액세스 하면 늦다. 가능한 한 캐쉬에 액세스 하도록 하면, 전체적인 대기 시간은 확실히 줄어 들기 때문이다. 그런데 이 캐쉬는, CPU와 같은 속도로 동작하기도 하고, 그렇게 간단히 제조할 수 있는 것은 아니다. 유명한 Pentium CPU은 앞서 말한 바와 같이 캐쉬 용량은 8KB에 불과하다. 그 큰 요인의 하나로서는 더 이상의 용량으로 하려고 하면, 회로 규모가 너무 커져서 , 제조가 어려워지는 점이 있었다.

회로 규모가 커지면, 왜 제조가 어려워지는 것일까? 이것은 제조 공정의 문제와 관계한다. CPU 에 한정한 이야기는 아니지만, PC용 디바이스의 모두에게 관련되는 이야기이므로, 여기서 설명해 두고 싶다.

CPU는 실리콘의 원반「웨이퍼」(Wafer)의 위에서 배열되 제조된다. 이 웨이퍼로부터 1개 1개 잘라서, CPU을 만들어 낸다. 이 때, 이 1개 1개 잘라낸 CPU의 회로를「CPU 다이」(Die)이라고 부른다. 즉, 여기서 말하는「커지는 회로 규모」란, 다이의 회로 규모이다.

웨이퍼를 사용해 제조할 경우에는, 어느 정도의「결함」의 발생을 피할 수 없다. 결함의 원인은 제조상의 불균일이거나, 티끌이나 먼지의 종류와 그 외 다양하지만, 결함이 발생해 버린 곳의 다이는 기본적으로 사용이 되지 않게 된다.

그럼, 웨이퍼상에 있는 정도 결함이 있었을 때, die size의 크기에 의해서,1매의 웨이퍼로부터 나오는 다이의 수는 어느 정도 차이가 나올까?. 이것을 나타낸 것이 그림 6이다. 여기에서는 직경40mm 의 웨이퍼가 있어, 거기에 9개소의 결함이 생겼다고 가정하고, die size를3 ×3mm 모퉁이로부터6 ×6mm 모퉁이까지 변화시켰을 경우에, 잡히는 다이의 수를 나타냈다.



(그림 6)

die size가 작다면(예를 들면3 ×3mm), 결함이 있어도 거기에 영향을 받는 부분도 최소한이 된다. die size가3 ×3mm 이라고, 1매의 웨이퍼에서는 이론상 112 개의CPU 다이가 잡히지만, 결함으로 18개는 불량품이 되기 때문에, Yield로 불리는 제품 비율율(우량품 CPU 다이를 생산되는 비율)은 약84 %가 된다.

그런데 CPU 다이의 사이즈를 크게 해 나가면, 결함에 의한 영향이 커지고, 우량품의 생산 수가 줄어 들게 된다. die size를3 ×3mm →4 ×4mm (으)로 하면, 제품 비율은83 %과 별로 다르지 않지만, 원래1 매의 웨이퍼로부터 잡히는 수가 줄어 들므로, 우량품인 다이의 수는94 개→50 개로 거의 반감을 한다. 한층 더5 ×5mm (면적비2.8 배)가 되면, 우량품은23 개, 6 ×6mm (면적비4 배)라면 12개까지 줄어 들어 버린다. 이런 6 ×6mm 으

로 만든다면, 반이 불량품이라고 하는 레벨이다.

실제, 초기의 CPU 제품 비율은 무서울 정도 낮았다. Intel 386의 제품 비율은 0.5개/ 매, 즉 웨이퍼를2매 제조해야 겨우 1개 우량품을 생산한다고 하는게 믿기 어려운 상태였던 것이다. 이것들은 제조 기술이나 제조 환경을 개선하는 것으로 점차 양호한 숫자가 되어 온 것이지만 우선 제품 비율을 높게 하기 위해서는, die size는 작으면 작을수록 유리하다고 하는 것이 된다.

サイズ	ダイ総数	良品ダイ数	理論上の製造原価	実際の製造原価
3×3	112	94	44.6ドル	53.2ドル
4×4	60	50	83.3ドル	100.0ドル
5×5	32	17	153.6ドル	294.1ドル
6×6	24	12	208.3ドル	416.7ドル

표1

더해 말하면, die size가 작은 만큼 CPU의 원가는 내려져 가는 것은 당연한 이치이다. 1매의 웨이퍼를 제조하기 위한 원가 = 실리콘의 원반 그 자체의 가격 + 그 위에 CPU 의 회로를 적층하기 위해서 필요한 가격이라고 하는 의미다. CPU 1개의 원가는 어떻게 되는지, 그림6 과의 관계로부터 이것을 정리한 것이 표1 이다.「이론상의 제조원가」란, 전다이가 우량품이었던 경우의 가격이고,「실제의 제조원가」란, 도6 의 조건으로 우량품 다이의 수를 고려한 가격이다.

이것이, 캐쉬 용량을 늘릴 수 없는 이유이다. 무엇보다, 제조 기술은 계속 끊임 없이 진화하고 있으므로, 캐쉬는 그림7 ~10과 같이 흘러 나와 점점 CPU 다이에 들어오게 되고 있다. 2005년 현재 유통하고 있는 CPU에 있어서의, 레지스터, 캐쉬, 메인 메모리의 관계는 그림 10과 같은 느낌이다. 그림 5에서 나타내 보인 이미지와 관련 지어 설명하면, L2 캐쉬(창고)가 CPU 다이(저택)안에 세운 느낌이다. 덧붙여서, 그림 10은 한층 더 진화했다. 기본적으로, 2005년 가을 시점의 CPU들은 그림 10의 구조라 생각해도 좋다. L1 캐쉬도 L2 캐쉬도 CPU의 안에 놓여져(지연시간은 다르지만)CPU (와)과 같은 동작 클락으로 동작하게 되어 있다.

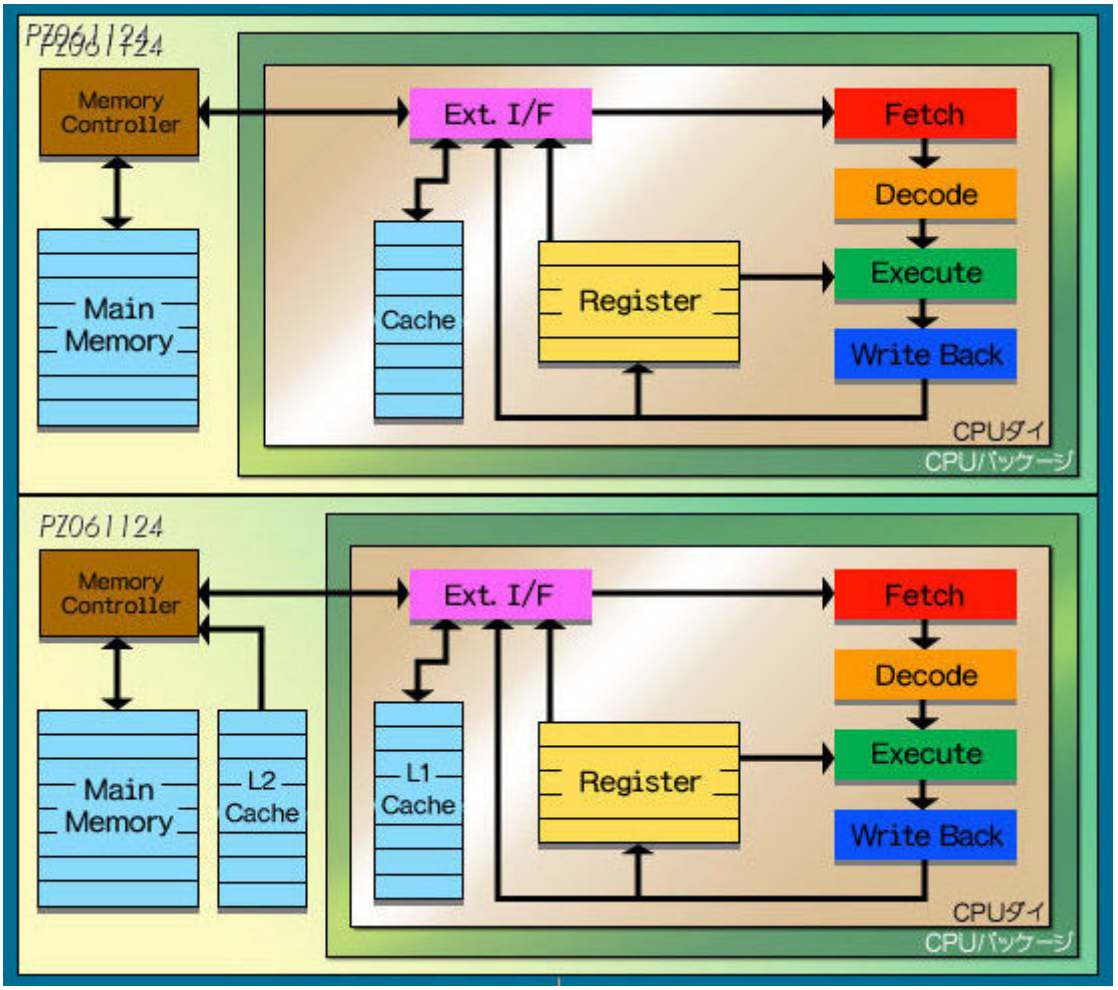

PZ061124
Memory Controller
Main Memory
Ext. I/F
Cache
Register
Fetch
Decode
Execute
Write Back
CPUダイ
CPUパッケージ
PZ061124
Memory Controller
Main Memory
L2 Cache
Ext. I/F
L1 Cache
Register
Fetch
Decode
Execute
Write Back
CPUダイ
CPUパッケージ

그림 7,8

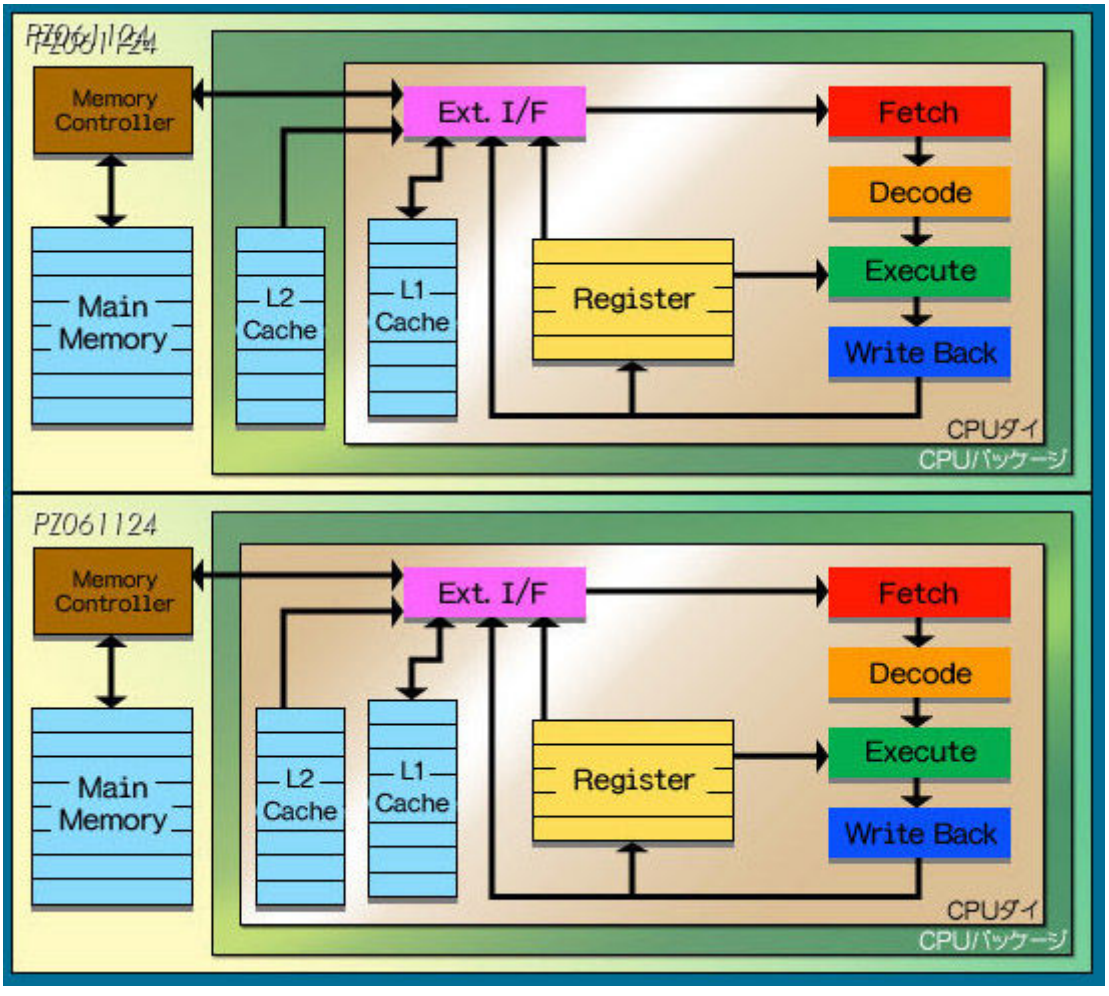

PZ061124
Memory Controller
Main Memory
L2 Cache
L1 Cache
Ext. I/F
Register
Fetch
Decode
Execute
Write Back
CPUダイ
CPUパッケージ
PZ061124
Memory Controller
Main Memory
L2 Cache
L1 Cache
Ext. I/F
Register
Fetch
Decode
Execute
Write Back
CPUダイ
CPUパッケージ

그림 9,10

CPU 의 성능 향상과 밀접하게 결합되는 제조 기술

한마디로 제조 기술이라고 말했지만, CPU(또는 그래픽 칩) 제조 기술은「프로세스」라는 말과 끊을래야 끊을 수 없는 관계에 있다.

「0.13 μm 프로세스」(0.13 마이크로 미터 프로세스)라고 하는「~프로세스」라고 하는 표기는 경제지등에서도 매우 잘 이용되므로, PC의 구조에 자세하지 않은 사람이라도, 한 번 정도는 본 것은 아닌가.

이「~프로세스」라고 하는 것은 한마디로 하면, 반도체를 제조할 경우에 이용되는 배선의 폭리 0.13 μm 와「0.13 ×10 의 마이너스6 승미터」라고 하는 길이를 나타낸다. 10의 마이너스9 승을 n(나노)로 나타내 보일 수 있기 위해, 130nm (와)과 표기하는 경우도 있다.

표2는 Pentium ~Pentium 4 까지,Intel 의 제공해 온 4 종류의 CPU로 어떻게 동작 클락을 올려 왔는지를 집계한 것이다.예를 들면Pentium은 최초로 0.8 μm BiCMOS하는 프로세스를 사용해, 최대 66MHz 동작하고 있었다. 옛날 그리운 이야기를 하면, 이것이「P5 」코어라고 하는 녀석이다. 그것이,0.09 μm (90nm) CMOS 프로세스의「Prescott 」코어에서는,3.80GHz 까지 달하고 있다는 것이다.

プロセスルール / 製品ライン	0.8μm BiCMOS	0.6μm BiCMOS	0.35μm CMOS	0.25μm CMOS	0.18μm CMOS	0.13μm CMOS	90nm CMOS
Pentium	66MHz (P5)	100MHz (P54C)	266MHz (P55C)				
Pentium Ⅱ			333MHz (Klamath)	450MHz (Deschutes)			
Pentium Ⅲ				600MHz (Katmai)	1.13GHz (Coppermine)	1.40GHz (Tualatin)	
Pentium 4					2GHz (Willamette)	3.46GHz (Northwood)	3.80GHz (Prescott)

※動作周波数の下にある括弧書きはCPUコアの開発コードネーム

표2

왜, 프로세스가 작아지는(전문적으로 말하면「미세화한다」) 것이 동작 클럭을 향상하는 것일까.

이유는 두 개 있다. 우선 배선폭을 미세화하면, 배선 경로 자체를 짧게 할 수 있기 때문이다. 예를 들면 그림 11과 같은 배선이 있다고 하자. 여기에서는, A 지점과 B 지점이 초록의 띠로 나타내 보인 것 같은 배선으로 연결되고 있어 1 단위의 길이를 도내에 나타낸 것 같은 사이즈로 규정하면, 배선의 길이는24 단위분이 된다.

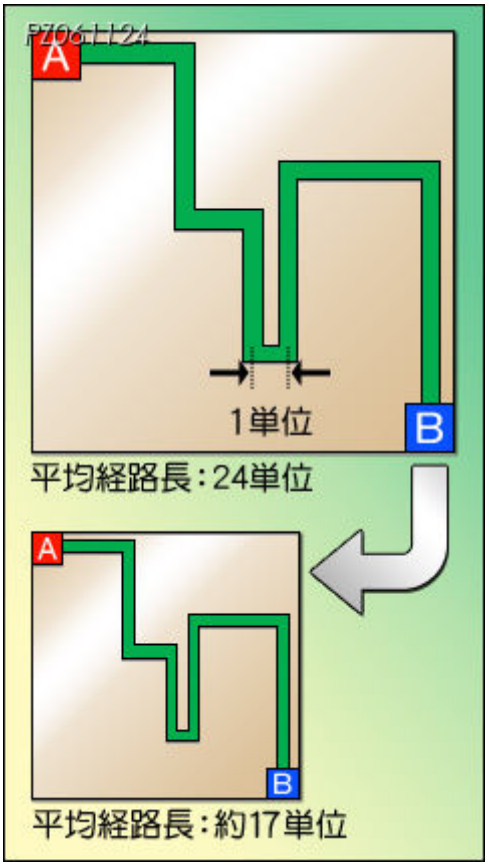


그림11

그럼, 이것을 미세화해 보자. 그림 11의 아래 쪽은, 위를 70%에 미세화해 본 것이다. 이 때 중요한 것은, 배선의 형태는 변하지 않아도, 사이즈가 작아진 덕분에, 배선장은 약 17단위(엄밀하게는 16.8 단위) 정도까지 짧아지는 것이다.

일반적으로 전류(전자)의 속도는, 동배선의 경우 광속의 3 분의 1 정도=시속 10만km 정도라고 말해진다.시속 10만km 그렇다고 하면 몹시 빠르다고 생각되지만, CPU 안에서는 유감스럽지만 그렇지 않다. 시속10 만km 로 진행되는 것은, 1ns (n =10 의 마이너스9 승) 사이에 동배선 10cm 밖에 진행하지 않는 것을 의미하는 것이다. 1장에서 설명한 것처럼, 이 1ns 라고 하는 것은 동작 클락 1GHz에 있어서 1 클럭 분의 시간이니까, 수 GH로 동작하는 최근의 CPU으로는, 그야말로 수cm 분 밖에 되지 않는다. 그런 만큼, 배선장은 가능한 한 짧게 하고, 배선안을 진행하는 시간을 가능한 한 짧게 해야 하기 때문에 프로세스를 미세화하는 것이다.

또 하나의 장점은, 트랜지스터 자체의 고속화이다. 조금 어렵지만, 대충 설명하자면, 최근의 CPU에 사용되는 CMOS란, MOS FET(Metal-Oxide Semiconductor/Field Effect Transistor)이라고 하는 트랜지스터를 페어로 해 이용하는 것이다.이FET (은)는,G (게이트),S (소스),D (드레인)(이)라고 하는 세 개의 단자를 가져, G - S 사이에 전압 (그림12 안에서는 푸른 점선 화살표)를 걸면, 거기에 응해 S - D 사이에 전류(그림12 안에서는 붉은 실선 화살표)가 흐른다고 하는 기능을 한다(그림12 좌측). 즉, 입력「전압」으로 출력「전류」를 컨트롤 할 수 있는 것이다.

이 FET가 어떤 형태로 실리콘상에 구성되어 있는가 하면, 대략적으로 나타내 보이면 그림 12 우측과 같은 이미지가 된다.

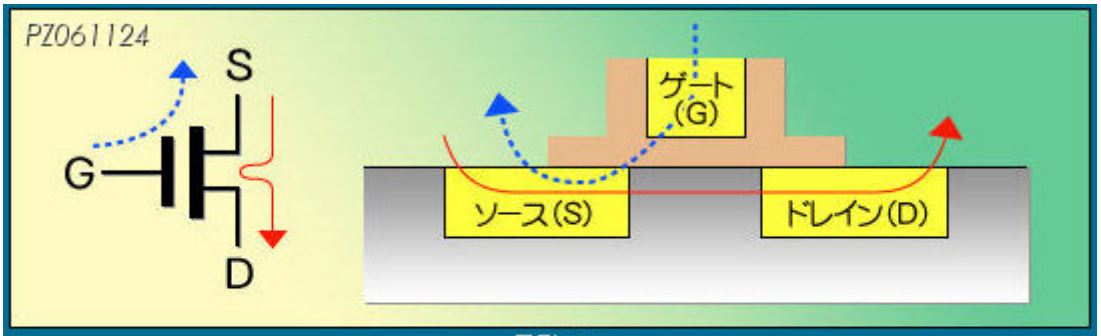


그림12

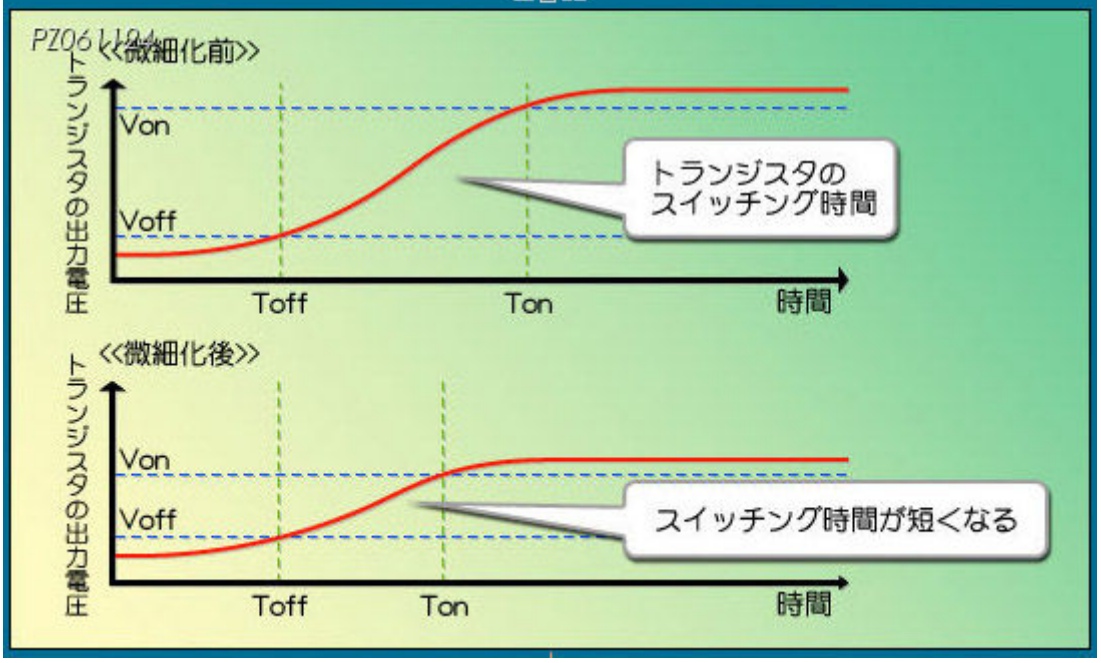


그림13

이전에는 트랜지스터의 물리적인 치수가 컸으니까, 트랜지스터도 스위칭 전압(온 상태와 오프 상태의 전압의 차이)이 큰 것이 필요했다.(라고 할까), 어쨌든 낮은 전압에서는 만족하게 동작하지 않았다. 트랜지스터의 전압은, 그림 13과 같이, 완만하게 변화하는 것이므로, 오프 상태(전압이Voff 보다 밑돌고 있는 상태)로부터 온 상태(전압이Von 보다 웃돌고 있는 상태)러 하려면 , 일정한 시간이 걸린다. Toff와 Ton의 사이의 시간, 이것ㅇ,ㄹ「트랜지스터의 스위칭 시간」이라고 불리는 것이다. 이 스위칭 시간을 단축하려면 , 급속히 전압을 시작하면 좋게 되지만, 신호가 안정되지 않는다든가, 소비 전력이 증가한다든가 말한 문제도 있고, 그래 간단하지는 않다.

하지만, 프로세스를 미세화하면, 트랜지스터의 스위칭에 필요한 전압이 내린다. 트랜지스터의 치수가 물리적으로 작아지기 때문에, 그만큼 고전압을 걸지 않아도 동작하게 된다. 전압 자체가 내리면, 완만하게 일어서는 경우여도 스위칭 시간은 큰폭으로 줄일 수 있기 때문에, 이것에 의해서 스위칭 속도(일정시간의 사이에 몇회 온/ 오프를 반복할 수 있을지)가 증가해 이것은 그대로 CPU의 동작 속도 상승으로 연결된다.

사실, 프로세스의 미세화가 가져오는 효과로서 제일 큰 것은, 이 트랜지스터의 고속화이기도 하다. 게다가 전압의 저하는 소비 전력의 저하와 이콜 하기 때문에, 어느 시기에는 프로세스의 미세화에 의해서, 「동작 클럭이 올라, 캐쉬 용량도 증가하면서, 소비 전력은 내린다」라고 하는 도식이 전개했던 것이다.

표3은 Pentium II ~4 의 특징을 집계한 것이다.「TDP 」(은)는「Thermal Design Power 」의 약어로,「열설계 전력」이라고 불린다. 간단하게 말하면, 기기 설계자에 대해서「이 발열량이 있다고 생각하세요」라고 하는 지침이 TDP 이다. Pentium III라면, 31 ~35W 의 전구가 PC에 들어가 있는 것이니까, 이것을 제대로 냉각해 주세요라고 하는 것으로 있다. 실제의 CPU 소비 전력 보다 약간 큰 값이지만, 그런데도 하나의 기준으로는 된다. 적어도 Pentium III 세대에서는, 동작 클락이 배 가깝게 되어도 TDP는 거의 변하지 않다든가, 동작 클락을 올

려도 소비 전력은 오히려 내리는 등, 매우 좋은 성적을 남기고 있는 것을 알 것이다.

P7061124	プロセスルール	トランジスタ数	動作周波数	TDP
Pentium Ⅱ	0.35μm	750万	333MHz	33.6W
	0.25μm	750万	450MHz	27.1W
Pentium Ⅲ	0.25μm	950万	600MHz	34.5W
	0.18μm	2800万	1.13GHz	35.5W
	0.13μm	4400万	1.40GHz	31.2W
Pentium 4	0.18μm	4200万	2.0GHz	75.3W
	0.13μm	5500万	3.4GHz	89W
	90nm	1億2500万	3.8GHz	115W

표3

최근에는 이러한 프로세스의 미세화에 의한 개량이 잘 되기 어렵게 되고 있다(Pentium 4이 그 좋은 예이다). 반대로 AMD의 Athlon 64같은 경우 90nm 프로세스를 사용하면서 자꾸자꾸 소비 전력을 내리는 것에 성공하는 등으로 향후에 다양한 개량이 전망되고 있는 것이 이 분야이다.

CPU 의 최신 트랜드「멀티 쓰레드」「멀티 코어」

최근, CPU를 둘러싼 이야기로 거의 확실히 나오는 멀티 코어 에 대해서도 설명해 두자.

또 다시 예를 들어, 집사 아래에 10 사람의 가정부가 있고,1 팀이 되어 있다고 한다. 일년에 한번 있는 대청소 때에는 10 사람이 풀로 활약해 저택안의 창문를 일제히 청소 하지만, 해마다 연중 바쁜 것은 아니라서, 평상시의 창 닦는 요원은 5 ~6 사람도 있으면 충분히로 하자. 이 때 남는 4 ~5 사람은 놀고 있는 것으로, 쓸데없게 급료를 지불하게 된다. 그러나, 그렇다고 해서 가정부의 수를 줄이면, 만일의 경우에 완전히 손이 부족하게 되기 때문에, 인원 삭감만은 절대로 용서되지 않는다.

거기서 발상을 전환해,「다른 청소 작업도 함께 실시한다」것으로 한다. 평상시는 가정부가 남아 있으니까, 그"잉여 전력"을, 마루 청소에 배분하면 된다. 그래서, 창문 청소가 일시적으로 바빠졌을 때는, 마루 청소 요원을 일단 귀환시켜 작업시켜, 반대로 마루 청소가 일시적으로 바빠졌을 때는, 창 닦는 요원을 여러명 마루 청소에 돌린다. 이렇게 하면, 모두가 일제히 창문 청소를 하고, 또 모두가 일제히 마루 청소를 하는 것보다도, 효율은 꽤 좋아진다. 이 방식의 경우, 동시에 바빠져 버렸을 때에는 대응할 수 없다고 하는 약점은 있지만, 그러한 사태가 좀처럼 생기지 않으면, 대체로 잘 되갈 것이다.

이와 같이, 복수의 처리(여기에서는 창문과 마루 청소이지만, 먼지청소 하는 도중도 있을 지도 모르다)를 1개의 CPU(여기에서는1 사람의 집사와10 사람의 가정부)로 관리하는 것을「multi-thread」(Multi Thread) 처리라고 한다. 멀티 쓰레드는 창 닦는데에 걸리는 시간이, 단일 처리를 하는 1개의CPU가 관리하는「싱글 스레드」(Single Thread)로 불리는 상태와 비교해서 큰 차이 없다. 그러나, 동시에 남은 연산기로 먼지철소 하는 도중도 실시하므로, 청소 토탈의 처리 시간은 큰폭으로 단축된다(=처리 능력이 큰폭으로 향상한다)라고 하는 구조다.

실제로 최근의 CPU의 동작을 생각하면, 복수의 처리가 동시에 실행되는 예는 드물지 않다. 그림 14를 보면, 게임 플레이 도중 Windows는 항상 백그라운드에서 움직이고 있고, 게다가 네트워크나 다양한 드라이버류가 필요에 따라서 가동한다. 게임은 당연 Windows 위에서 움직이고 있기 때문에,「게임만을 하고 있다」라고 하는 것은 (외형으로는 올바를지도 모르지만) 현실에는 그만큼 정확한 표현은 아니게 된다.

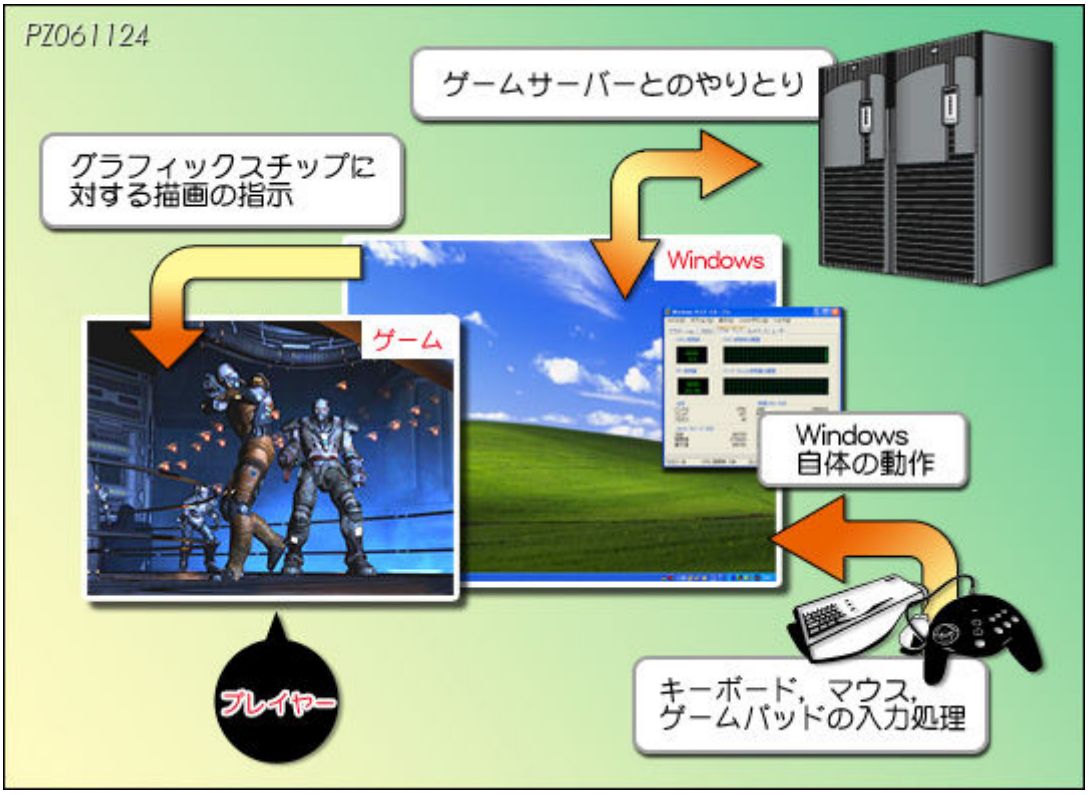


그림 14

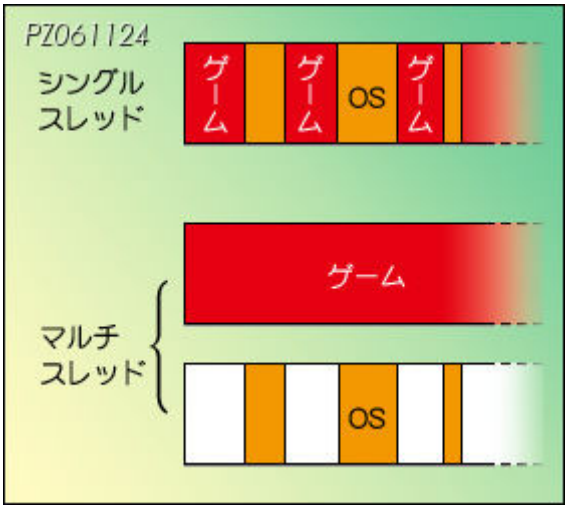


그림 15

예전의 이러한 복수의 처리는, 싱글쓰레드가 단시간 동안에 바꾸면서 행해지고 있었다. 이미지로써는 그림 15 와 같은 느낌이다. 실제로, 이 처리 변환에 필요로 하는 시간은 극히 짧기 때문에, 지연되는 시간을 유저가 체감 할 수 있는 기회는 거의 없지만, 그런데도, 그만한 시간이 드는 것은 사실이다. PC 전체적으로는 약간 성능이 떨어져 결과적으로, 게임의 체감 속도는 내림세가 된다. 한편 멀티 쓰레드 라면, 그림 15 아래와 같은 작업의 할당이 가능하게 되어, 게임의 체감 성능은 약간 향상하게 되는 것이다.

이러한 처리를 나타내는 용어는, 쓰레드 외에도「프로세스」「작업」「태스크」가 있다. 뭐라고 이야기하여 들었던, 물었던 적이 있는 사람이 많을 것이지만, 실은 각각 제대로 의미가 있다. 가정부의 일로 생각하면, 쓰레드는 구체적인「청소의 내역」이라고 하는 것이지만, 태스크라면「청소」「세탁」「요리」라는 구분이 된다.「작업」「태스크」도 넣고, 가정부의 일로 생각하면, 대체로 이하와 같은 이미지다.

태스크 :「저택과 창고를 돌본다」
작업 :「저택을 돌본다」「창고를 돌본다」
프로세스 :「저택의 청소」「저택의 세탁」「저택의 요리」「창고의 청소」등

스렛드 :「청소」의 내역,「마루 청소」「창 닦이」「먼지 청소」등

즉, 큰 목표가 태스크 이고, 작업, 프로세스, 스렛드라고 하위 개념으로 세분화되어 가는 것이다.

싱글쓰레드 대응 CPU의 경우「먼지 청소」와「마루 청소」는 동시에는 해낼 수 없다. 따라서「무서울 기세로 먼지 청소를 해, 그것을 마치고 나서 굉장할 기세로 마루 청소를 한다」또는,「1분간 마루청소하고, 다음의1 분간은 먼지 청소, 그 다음의1 분간은 또 마루 청소 를 반복한다」로 어느 쪽이던지 시간은 같을 것이다.
그런데 멀티 쓰레드 대응의 CPU는,「먼지청소」와「마루 청소」를, 문자 대로 동시에 실행할 수 있다.어느 가정부가 먼지 청소 하고 있는 동안에, 다른 가정부가 마루 청소를 하는 이미지이다.

단지 하나 문제가 있는데, 멀티 태스킹이나 multi job이라고 한 큰 레벨에서는, 프로그램을 복수 동시에 움직일 뿐이지만, 멀티 프로세스나 multi-thread가 되어 지면, 움직이는 프로그램 안에서 명시적으로 처리가 세분화되어 있지 않으면 안 된다. 요컨데, multi-thread를 의식해 만들어지지 않은 프로그램은, 싱글스 레드로 움직인다. 그래서 멀티 쓰레드 지원하는 CPU를 가져와도 처리는 빠르게 안 된다.여러가지 프로그램이 multi-thread 대응이 되어야만, 멀티 쓰레드 CPU의 진가가 발휘된다.

멀티 쓰레드 지원 CPU 의 가장 일반적인 예로서는,「Hyper-Threading 테크널러지」로 확실히 복수의 가정부가 동시에 움직이도록(듯이), 복수의 스렛드를 동시에 처리하고 있다.

하지만 이런 경우, Pentium 4의 효율이 매우 나쁘다고 하는 문제가 있다. 집사1 사람, 가정부10 사람의 예로 설명하면, Pentium 4의 경우, 아무것도 일하지 않는 가정부는 항상 3 사람 정도, 나머지의7 사람은 게으른 잠을 탐내고 있다. 여기서HT 테크놀러지를 사용하면, 항상 5 사람 정도 일하게 된다.「그런데도 아직5 사람은 자고 있는지!」라고 공격이 들어갈 것 같지만, 전원을 동시에 일하게 한다는 것은 여러 가지 있어 몹시 어렵다고 이해받으면 좋을 것이다.

이 생각을 한층 더 진화시킨 것이「듀얼 코어」, 즉 2개의 CPU 코어를 동시에 일하게 하는 구조이다.「집사가 2사람, 가정부20 사람」의 구성에 파워업 해, 2팀으로 나누어지고 작업을 진행시킨다.말할 필요도 없이, 이것은 CPU 2개 있는 것과 같은 것이므로, 동시에 해낼 수 있는 일량은2 배가 된다.

단지, HT 테크놀러지와 달리, CPU 코어 1개 만큼 효율이 오르는 것은 아니다(20 사람의 가정부는,10 사람씩 각 팀에 배치되어 인원의 이동은 없다), 효율의 나쁨 자체는 해결되지 않는다. Pentium 4의 코어를 2개 탑재한 Pentium D 경우, HT는 이용할 수 없기 때문에 20 사람 있는 가정부 가운데, 일해 주는 것은3 ×2 = 6 사람뿐이고, 나머지14 사람은 자고 있는 계산이 된다. 그런데도 HT를 이용한 Pentium 4의 5사람 보다는 조금 빠르다고 하는 것이다.

무엇보다 최대의 문제는, CPU를 사실상 2개 탑재했더니「조금 빠르다」정도에 지나지 않는 것이지만.

CPU 의 주변에도 시스템 전체의 성능에 영향을 준다

L2 캐쉬가 CPU 안에 짜넣어져 간 것처럼, CPU를 둘러싸는 주변 환경도 시대와 함께 진화하고 있다. 이번은 마지막으로, 이 부분을 간단하게 살펴 보자.

초기의 시스템은, 그림 16과 같은 구성을 취하고 있었다. CPU에서는「I/O 버스」(Input/Output Bus)(으)로 불리는 버스 ※ 에의 인터페이스만이 준비되어 버스의 끝에 메모리나 주변기기가 함께 매달려 있었던 것이다. 당시는 CPU의 동작 클럭이 낮았으니까, 이것으로 충분히 늦지 않은 상황이었다. 그 후 앞서 말한 바처럼 - CPU의 고속화에 수반하고, 메모리도 고속화할 필요가 나왔다. 그러나, 주변기기는 따로 고속화할 필요가 없었다.

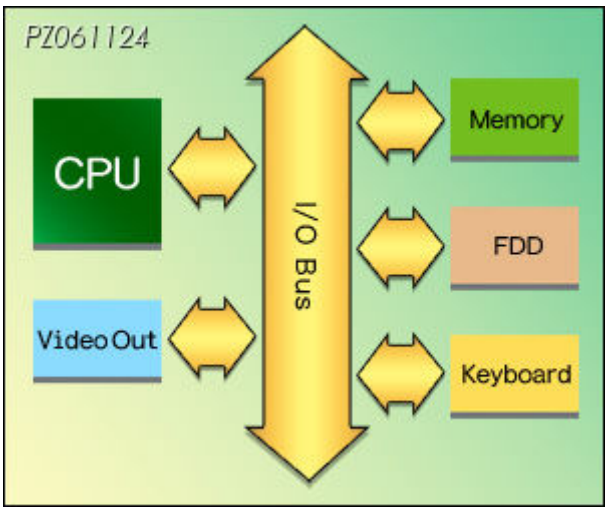


그림 16

여기서 등장한 것이 고속버스다. 현재의 PC 기본으로 된「IBM PC/AT 」라고 하는 PC의 기능을 한덩어리로 한「칩 셋」(Chipset)이라 불리는 구조에, 보다 고속의「시스템 버스」(System Bus)를 제어하는 콘트롤러가 짜넣어진 칩 셋은, 이 시스템 버스 컨트럴러를 축으로 한 것이 되었다.

시스템 버스의 대표격은, 현재도 사용되고 있는 PCI 버스(Peripheral Component Interconnect Bus)이다. PCI 버스 이후의 칩 셋은 기본적으로 2개 팁 이상이 되어, CPU와 직접 접속하는 편을「노스 브릿지」(North Bridge), 노스 브릿지를 개입시켜CPU (와)과 접속하는 편을「사우스 브릿지」(South Bridge)(이)라고 부르는 것이 일반적 구성이며, 그림 17과 같다. 이것이 현재 사용되고 있는 칩 세트의 원형이 되고 있다.

노스 브릿지는 캐쉬나 메인 메모리 그래픽 카드 등 고속의 디바이스를, 사우스 브릿지에는 HDD 나 USB ,LAN ,키보드/ 마우스 등의 상대적으로 저속인 디바이스를 각각 접속하는 구조다. 덧붙여서 그림 17에서 보면, 칩 세셋과 캐쉬가 접속되고 있지만, 이 사양이 벌써 쓸모없게 되고 있는 것은 앞서 말한 바와 같다. 그리고 노스 브릿지와 사우스 브릿지의 사이는 PCI 버스로 접속되고 있지만, 이것도 현재는 보다 고속의 버스가 이용되고 있다. 버스에 대해서는 다음 회에서 설명할 것이니..일단은 그러한 것이라 생각해 준다면 좋다.

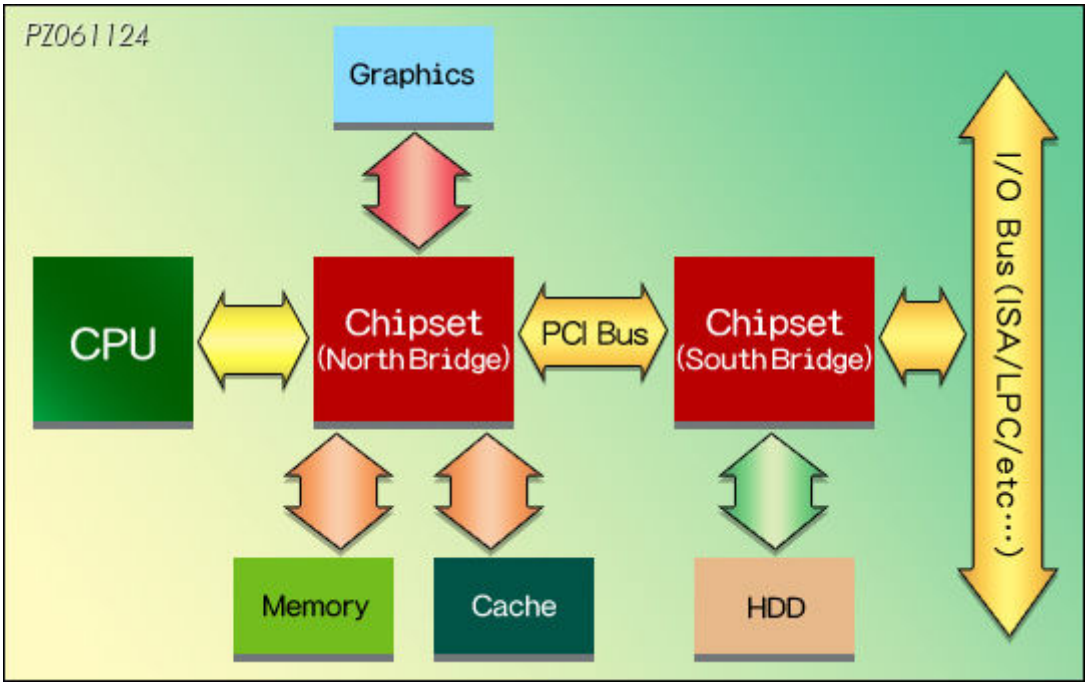


그림 17