T1: Métricas

Sostenibilidad:

- Economia: Reducion de costes => deslocalizacion empresas,
- Ambiental: Construccion de productos y destrucion, etc...
- Social: Enfermedades en las personas, perdida del contacto.
- Coste Humano: Problematica del Coltan -->Problemas humanos.

  Economia, Social y Ambiental => Solucion Sostenible.

Evaluación del Coste

$$CosteTotal = \frac{(CosteDie + CosteTest + CosteEmpaquetado)}{Yieldfinal(test)}$$

$$CosteDie = \frac{CosteWaffer}{DiesPerWaffer * DieYield}$$

$$DiesPerWaffer = \frac{\pi * (diametro/2)^2}{DieArea} - \frac{\pi * Diameter}{\sqrt{(2 * DieArea)}}$$

$$DieYield = WafferYield * (1 + \frac{DefectosPerArea * DieArea}{\alpha})^{-}a$$

Ley De Moore

- Observacion Economica, no Tecnologica.
- El numero de transistores (que de forma economica) se pueden integrar en un circuito se duplicara cada X tiempo.

Otras medidas de Coste

Latencia: Tiempo que transcurre entre la solicitud de un dato y la disponibilidad del mismo

Ancho de banda: Numero de Bytes transmitidos por unidad de tiempo.

Productividad: Es el trabajo realizado por unidad de tiempo.

Tiempo Respuesta: Es el tiempo necesario para procesar una solicitud.

Rendimiento

$$\frac{1}{Rendimiento} = TiempoEjec = Instruciones * CPI * TiempoCiclo$$

Los factores que afectan al tiempo de ejecucion NO son independientes.

$$Speeedup = \frac{Ta}{Tb}$$
$$TantPerCent = (\frac{Ta}{Tb} - 1) * 100$$

Si > 1 --> B es mas rapido que A

Si < 1 --> A es mas rapido que B

Existen otras metricas para comparar:

MIPS: Millones (10^6 Instruciones) por segundo.

MFLOPS: Millones de Floating Point Inst por segundo.

Consumo

$$Potencia = Conmutacion + CorrienteFugas + CorrienteCortocircuito$$
$$Potencia = I * V = Watios$$
$$Energia = P * T = Julios$$
$$PotenciaConmutacion = C * V^2 * freq$$
$$EnergiaConmutacion = C * V^2$$
$$PotenciaFugas = I_{Defuga} * V$$
$$EficienciaEnergetica = \frac{rendiminento}{watio} = \frac{1}{EnergiaConsumida}$$

Fiabilidad

$$ProbFallo = 1 - e^{-\lambda * \Delta t}$$
$$\lambda = \frac{1}{MeanTimeToFail}$$
$$Disponibilidad = \frac{MTTF}{MTTF + MeanTimeToRepair} = \frac{MTTF}{MeanTimeBetweenFails}$$
$$\frac{1}{MTTF_{sistema}} = \frac{1}{MTTF_1} + \frac{1}{MTTF_2} \cdots$$

T2: LM

Conceptos Basicos

Tipos Basicos, En little Endian

RANGOS:	Byte(8 bits)	Word(16)	LongWord(32)
Naturales	0 a 255	0 a 65.535	4.294.967.215
Enteros(Ca2)	-128 a 127	-32.768 a 32.767	-2.147.483.648 a 2.147.483.647

1 byte = Char

2 byte = Short

4 byte = Int, *pointer,

8 byte = Float, Double

12 byte = Long Double

Modos Direccionamiento

- Registros
  - Registros(LongWord(4 byte)): %eax, %ebx, %ecx, %edx, %esi, %edi
  - Registros(Word(2 byte)): %ax, %bx, %cx, %dx
  - Registros(Byte): %ah, %al, %bh, %bl, %ch, %cl, %dh, %dl
  - Registros Control: %eflags, %esp - %ebp (subrutinas), %eip (PC)
- @[Memoria] = Rbase + Rindice * Scale(1, 2, 4 o 8) + Offset => O(Rb, Ri, S)
- Inmediato = $12, etc..

Tipos de datos Estructurados

- Vectores: "Seq de datos iguales almacenados en memoria".

  v[i] -> @v + i * sizeof(tipo).
- Matrices: Vectores almacenado por filas.

  A i, j : @A + (i* NumCols + j) * sizeof(tipo)
- Matrices 3d: Se almacenan tmb en posiciones consecutivas.

  3d i, j, k: @3d + (i * Numero_cara2 * Numero_cara3 + j * Numero_cara3 + k) * sizeof(tipo)
- Las instrucciones multimedia sirven para operar una gran cantidad de operaciones sobre tipos estructurados.
- Los structs como tal, no son reconocidos, el programador debe mantenerlos manualmente.

Alineaminento de Datos (linux 32 bits)

Para asegurar que los sistemas de cache i paginación funcionan correctamente y para poder hacer acesos a memoria por longword o quadword, los datos deben alinearse. El compilador es el que inserta "espacios en blanco". Por tanto, "una direccion debe ser múltiplo de :

- char alineado a 1-byte (@ cualquiera)
- short alineado a 2-bytes (@ ...0)
- int alineado a 4-bytes (@ ...00)
- puntero(4 bytes), double(8 bytes), Long double(12 bytes) alineado a 4-bytes

Las estructuras, debe cumplir la restriccion del elemento maximo(maxima alineacion) que contien sus campos, tambien su @ inicio lo tiene que cumplir. El orden de los campos de un struct pueden hacer que este ocupe mas.

GESTIO DE SUBRUTINAS

En C-linux 32 bits, los parametros se pasan por la pila, **de derecha a izquierda**.

- Vectores y matrices siempre se pasan por &.
- Los structs se pasan por valor siempre.
- Char(1byte) ocupan 4 bytes y Short(2bytes) ocupan 4 bytes.
- Las variables locales se alinean en la pila como si fuera un struct. El tamaño de variables locales debe ser multiplo de 4.
- **%ebp, %esp se salvan siempre implicitamente.**
- **%ebx, %esi, %edi se han de salvar si son modificados por la rutina**.
- **%eax, %ecx, %edx se pueden modificar dentro de la subrutina, el llamador debe salvarlos.**
- **Los resultados se devuelven por %eax.**
- La pila siempre debe estar alineado a 4.

Bloque de activacion - Ejemplo de Subrutina

1. Paso de Parametros y llamada

   Se colocan en la pila de derecha(primeros) a izquierda, al hacer push el %esp se desplaza 4 hacia arriba(%esp - 4). Los registros que utilizamos (%eax, %ecx o %edx) deben ser salvados antes de llamar. Por ultimo, se invoca la subrutina, al hacer call %esp = %esp - 4 y se guarda la @ret.

2. Enlace dinámico y * al bloque de activacion

   Push de %ebp a la pila y %ebp se coloca donde %esp.

3. Variables locales y salvar estado del llamador

   Agrandamos la pila, subiendo el %esp para hacer caber las variables locales, en el orden que se declaran, de arriba a abajo. Los registros que necesitaremos utilizar (%ebx, %esi o %edi) los guardamos en la pila; los otros pueden ser modificados sin problema.

4. Cuerpo de la subrutina y retorno

   Se ejecuta el cuerpo de la subrutina, teniendo en cuenta el bloque de activacion. El resultado a retornar siempre debe colocarse en %eax.

5. Restaurar estado de registros y eliminacion de variables locales.

   Pop de los registros que hayan sido utilizados (%ebx, %esi o %edi), para las variables locales, bajamos el %esp (%esp + x). Por ultimo restauramos el %ebp y hacemos ret.

6. Recoger/usar el resultado

   Para la rutina padre, el resultado estara en %eax;

Gestion de Registros en Subrutinas

- Registros %eax, %ecx i %edx (registros no seguros)

  Se pueden modificar dentro de una subrutina --> El padre debe guardarlos al invocar.

- Registros %ebx, %esi, %edi (registros Seguros)

  Si una subrutina quiere modificarlos, debe guardarlos --> El padre no debe guardarlos al invocar.

- Registros %ebp, %esp

  Registros para la gestion de la pila.