임베디드 어플리케이션 분석

1. 스택
4. 어느 지점에서 특정 지점을 점프해야하는 상황이 생길 때 사용하면 유용하다.
5. 포인터의 크기는 cpu 가 레지스터에 몇 bit 를 제공하는냐에 따라 달라진다. 레지스터의 총 bit 수 만큼 데이터가 저장이 가능하고, 각각의 데이터에 해당하는 주소가 필요하다. 즉 데이터 수와 주소의 수는 1:1 이고, 결국 주소를 저장하는 포인터의 크기는 데이터 가짓 수에 따라 결정된다.
8. void (* signal(int signum, void (* handler)(int)))(int)
→ void (*)(int) signal(int signum, void (* handler)(int))
1) 반환 : void (*)(int)
→ void 를 반환하고 int 를 인자로 취하는 함수의 포인터
2) 함수명 : signal
3) 인자 : (int signum, void(* handler)(int))
→ int 형 데이터 signum, void 를 반환하고 int 를 인자로 갖는 함수 포인터 handler
15. Intel Architecture 는 CISC 이고 ARM Architecture 는 RISC 이다.
CISC 는 명령어의 종류를 다양하게 해서 소프트웨어를 간단하게 하지만, 명령어 하나당 여러 clock 을 거쳐야한다. 또한 명령어를 다양하게 만들기 위해서 집적도가 높아져서 발열이 심하다.
RISC 는 명령어를 최대한 간단하게 만들어져서 저전력이라는 장점이 잇지만, 소프트웨어가 복잡해지는 단점이 잇다.
19. int *p[3] : int 형 포인터를 3 개 저장하는 배열
int (*p)[3] : int 형 데이터를 3 개 저장하는 배열에 대한 포인터
→ 결국 다른 것이다.
22. 변수 : 데이터를 저장할 수 잇는 공간
23. 포인터 : 주소를 저장할 수 잇는 변수
24. 함수포인터 : 함수의 주소를 저장할 수 잇는 변수
25. branch 나 jmp 명령어를 만나면 파이프라인이 깨지게 된다.
26. 메모리는 속도가 빠른 순으로 레지스터-캐시-메모리-디스크 순이고, 가격은 역순이다.
27. c 언어의 기본 메모리는 stack, heap, data, txt 로 나누어 진다.
1) stack : 지역변수가 위치하는 영역
2) heap : 동적할당된 변수가 위치하는 영역
3) data : 전역변수 및 static 으로 선언된 변수들이 위치하는 영역, 초기화가 되지 않으면 0 으로 저장됨.
4) txt : machine code 가 위치하는 영역

28. 디버깅하면서 보는 변수의 주소는 실제 메모리상의 주소가 아니다. 이 가짜 주소들을 운영체제가 페이징 기법을 통해 memory table 과 연결시켜서 실제 물리 메모리 주소에 맵핑시켜주는 것이다.
30. gcc -g -o debug c 파일명 으로 디버깅을 한다.
32. gcc -g -o debug c 파일명 으로 하면 최적화를 해준다.
gcc -g -o0 -o debug c 파일명 으로 최적화를 방지한다.
33. 프로그램이 컴파일상의 오류는 없지만 논리적인 오류가 잇어서 이를 분석하기 위해서 사용함.
34. push %rbp // mov %rsp,%rbp // sub %rsp $-8 // pop %rbp
40. 강사님의 열정적인 강의에 감사드립니다. 질문에도 최대한 답변해주셔서 더욱 더 많이 발전한 것 같습니다.
앞으로 이번 한달 동안 햇던 것보다 좀 더 열심히 해보도록 노력하겠습니다!
앞으로 SW/FW 분야에 취업을 해서 실력을 더 키워나갈 생각입니다.

자료구조

7. AVL 트리는 삽입과 삭제를 할 때마다 트리의 회전여부를 검사하고 필요시 회전을 수행해야 하기 때문에 삽입과 삭제를 빈번하게 해야하는 분야에서는 성능이 저하된다. 반면에 삽입과 삭제를 거의 하지 않고, 검색을 많이 하는 분야에는 매우 좋은 성능을 보인다. 모든 트리 중에 AVL 트리가 검색 속도가 가장 빠르다.
Red Black 트리는 삽입과 삭제와 검색 속도를 모두 양호하게 만든 트리이다. 물론 검색속도는 AVL 트리보다는 느리다. 따라서 삽입과 삭제가 빈번하게 수행되는 분야에 적합하게 사용된다.
13. 불필요한 연산을 줄인다.