11\_25 컴퓨터 시스템 아키텍처 정리

141019 김연우

1. 어셈블리 조건문 연계
   1. 순환문 (loop)
      1. 조건부 점프 명령어. 점프 주소가 음수 값 (앞으로 가는 점프 문)
      2. 컴파일러는 Do – While 형태로 순환 문을 해석한다.
      3. 일반 While문이면, 맨 앞에 조건부 점프 문 하나 넣으면 됨.
      4. 진입 시점에서는 False인 경우를 점프. 반복 시점에서는 True인 경우 점프
      5. For문의 경우 일반 While문과 크게 다르지 않다. 초기값 세팅
      6. %eax는 리턴값 저장용
   2. Switch Case
      1. 점프 테이블 활용
      2. 점프 주소 배열을 이용하는 간접 점프 (포인터 배열: jmp \*Label(, %eax, 4))
      3. Case의 라벨은 배열의 인덱스로 변환하므로 값이 조밀하게 분포하는 경우에 적합
      4. 점프 테이블은
2. 프로시저 (함수)
   1. 스택
      1. 스택의 존재 이유
         1. CPU는 그저 연속된 어셈블리 코드를 처리할 뿐, 프로시저 단위를 알지 못함
         2. 프로시저는 인간이 코드들을 효율적 구조적으로 활용하기 위해 만들어진 것
         3. 독립적 구조로서 프로시저는 분리된 컨텍스트를 갖고 있어야 함. (컨텍스트 스위칭!)
         4. 함수가 호출(점프)이 되면, 호출된 함수의 컨텍스트를 사용, 리턴되면 본래의 컨텍스트로 돌아옴
         5. 이런 컨텍스트가 저장되는 공간이 스택.
      2. 스택의 구조
         1. 스택의 top 포인터(%esp)
         2. pushl, popl 명령어로 스택에 save/load & 스택 포인터 유지
         3. 스택 프레임 : 하나의 프로시저에 할당된 부분
         4. 스택 프레임(의 시작주소) 포인터(%ebp)
         5. 함수 호출/반환할 때 스택 레지스터 변경/저장
         6. 기존에 사용하던 레지스터 정보는 새로운 프레임의 상단에 Savaed Register에 저장한다.
         7. 프로시저(함수)의 인자는 호출한 쪽의 스택 프레임이 가지고 있다.
      3. 스택 활용 방식
         1. 호출하는 프로시저는, 뒤쪽 인자부터 저장. 맨 마지막에 리턴 주소저장.
         2. 호출된 프로시저는, 기존 프레임 포인터 값 저장.
         3. 새로운 명령어 : call, ret , leave
         4. Call : 현재 스택에 %eip (pc) 값을 저장, & esp ++;
         5. Ret : 저장된 이전 pc값을 pop해서 pc값 변경. (점프 노필요)
         6. Leave : 리턴하기 전에 리턴 값을 리턴 어드레스에 저장해둠