ELABORATO ASM

Laboratorio di Architettura degli Elaboratori 2014/2015

Mario Alessandro Barbara (VR 387794)

DESCRIZIONE DEL PROGETTO

È chiesto allo studente di sviluppare un programma che simuli il controllo per l'inclinazione dei flap di un Airbus A320 in base al numero e alla distribuzione dei posti a sedere dei passeggeri.
Il programma e' da scriversi in linguaggio assembly, con sintassi Intel x86.

Vigono le seguenti limitazioni: il numero di posti massimo dell'airbus e' di 180 passeggeri, disposti su 6 file (A, B, C, D, E, F).

Il programma deve essere in grado di riconoscere il codice di controllo di sicurezza inserito (come parametro all'avvio del programma oppure come input durante l'esecuzione del programma) e di prendere delle decisioni in base ad esso.
I codici di sicurezza riconosciuti sono I seguenti:

- “9 9 2” → codice della **Modalita' di Emergenza** del programma
- “3 3 2” → codice della **Modalita' Dinamica** del programma

Dovesse il codice essere errato, in seguito a 3 input errati consecutivi del codice, il programma entrera' in Modalita' Safe, terminando subito dopo.

Modalita' di Emergenza: il programma stampera' la modalita' in cui si trova e terminera'.

Modalita' Dinamica: il programma stampera' la modalita' in cui si trova e richiedera' in input il numero di passeggeri dell'airbus e la loro disposizione su 6 file (A, B, C, D, E, F). Dovessero gli input non essere coerenti fra di loro, il programma li chiedera' di nuovo. Una volta terminata la fase input della modalita', iniziera' una fase di calcolo: verranno calcolati i 4 bias in base alle seguenti 4 equazioni:

- Bias1 = (x/2) * k1 + (y/2) * k2
- Bias2 = (y/2) * k2 + (z/2) * k3
- Bias3 = - (y/2) * k2 - (z/2) * k3
- Bias4 = - (x/2) * k1 - (y/2) * k2

(con X=A-F, Y=B-E, Z=C-D e k1=3, k2=6, k3=12)

Le equazioni si possono facilmente semplificare in:

- Bias1 = (3/2) * (x + 2y)
- Bias2 = 3 * (y + 2z)
- Bias3 = - Bias2
- Bias4 = - Bias1

Dopodiche' i 4 bias verranno stampati e il programma terminera'.

Mario Alessandro Barbara (VR 387794)

VARIABILI UTILIZZATE E IL LORO SCOPO

All'interno del programma le uniche variabili (tutte globali) utilizzate sono stringhe di caratteri ascii, ad eccezione della variabile globale “buffer_clean”. Essa consiste in un long (numero a 32bit) che non verra' mai usato, esso e' necessario per evitare problemi con il buffer a 64bit durante la lettura di caratteri in modalita' 32bit, come viene eseguita in questo programma intel x86.

Le variabili stringa non sono accompagnate da alcuna variabile corrispondente alla loro lunghezza. Si e' preferito utilizzare una funzione che stampasse autonomamente le stringhe (“print_string”) tramite il passaggio del loro indirizzo in memoria. Per questo motivo tutte le stringhe terminano con il carattere nullo '\0'.
Inoltre, i nomi di tutte le variabili stringa sono in italiano. Questo le rende piu' facilmente differenziabili dalle funzioni del programma, che hanno tutte nomi in inglese.

Ecco una lista delle variabili:

- **buffer_clean** – long non utilizzato. Necessario per il funzionamento del programma
- **codice_errato** – stringa di errore stampata quando viene richiesto all'utente di reinserire il codice, dopo un tentativo errato
- **codice_pretty** – stringa stampata per rendere piu' visibile l'avvio di una delle 3 modalita' (dinamica/emergenza/safe)
- **codice_dinamico** – stringa stampata quando viene attivata la modalita' dinamica del programma
- **codice_emergenza** – stringa stampata quando viene attivata la modalita' di emergenza del programma
- **codice_failure** – stringa stampata quando viene attivata la modalita' safe del programma
- **inserire_totale** – stringa stampata per chiedere l'inserimento del numero di passeggeri sull'airbus
- **inserire_totale_err** – stringa di errore stampata per annunciare che il numero di passeggeri sull'airbus appena ricevuto eccede il massimo di 180
- **inserire_filesx** – stringa stampata per chiedere l'inserimento del numero di passeggeri sulle file A, B, C
- **inserire_filedx** – stringa stampata per chiedere l'inserimento del numero di passeggeri sulle file D, E, F
- **inserire_file_err** – stringa di errore stampata per annunciare che il numero di passeggeri delle file appena ricevuto eccede il massimo di 30 per fila
- **inserire_totali_diversi** – stringa di errore stampata per annunciare che il numero di passeggeri sull'aereo non coincide con il numero di passeggeri presenti sulle file
- **stampa_bias** – stringa stampata per fornire il risultato di uno dei 4 bias

Mario Alessandro Barbara (VR 387794)

- **stampa_remainder** – stringa utilizzata nel caso fosse necessario stampare un numero con cifre decimali (nel primo e quarto bias)
- **stampa_enter** – stringa stampata per andare a capo linea. E' stata usata per rendere piu' visibile l'output del programma

MODALITA' DI PASSAGGIO DEI VALORI FRA LE FUNZIONI

Tutte le variabili vengono passate esclusivamente per registro. Questa scelta e' stata fatta poiche', trattandosi di un progetto piuttosto ristretto, risultava piuttosto facile prevedere lo spostamento dei valori dei registri fra una funzione e l'altra. In un programma piu' grande questa strategia potrebbe essere sconsigliabile.

_start: la funzione iniziale non passa alcun valore. Si e' deciso di tenerla il piu' possibile slegata ad altre funzioni, per favorire la modularita' del programma.

check_ext_param:
→ la funzione non riceve parametri
← la funzione mette nel registro EDI il numero di volte (3) che i valori del programma andranno richiesti (dalla funzione “recheck”) nel caso fossero errati. Nel caso il numero di parametri passati all'avvio del programma fosse corretto, mette nei registri EAX, EBX ed ECX le 3 cifre che saranno verificate dalla funzione “params_check”

params_check:
→ la funzione riceve i 3 numeri del codice da verificare nei registri EAX, EBX, ECX
← la funzione non passa alcun valore di ritorno, ma preserva il valore del registro EDI

recheck:
→ la funzione riceve in EDI il numero di volte che deve continuare a reiterare, e lo decrementa ogni volta
← nel caso la funzione di lettura parametri “atoi_30” dovesse avere successo, la funzione riceve nei registri EAX, EBX ed ECX I 3 numeri da passare alla funzione “params_check” (chiamata subito dopo) affinche' siano verificati. Inoltre, il valore del registro EDI viene preservato

safe_mode:
→ la funzione non riceve parametri
← la funzione non passa alcun valore di ritorno

emergency_mode:
→ la funzione non riceve parametri
← la funzione non passa alcun valore di ritorno

dynamic_mode:
→ la funzione non riceve parametri
← la funzione non passa alcun valore di ritorno

print_bias:
→ la funzione riceve in EBX il valore del bias da stampare
← la funzione non passa alcun valore di ritorno

Mario Alessandro Barbara (VR 387794)

atoi_3:
→ la funzione riceve in EDI il numero di interi da leggere in input (1 o 3)
← la funzione mette nel registro EDX 1 nel caso di lettura corretta, 0 nel caso di lettura fallita. Nel caso di lettura corretta mette nei registri EAX, EBX ed ECX i 3 numeri letti, oppure nel registro EAX il numero letto (a seconda del valore passato alla funzione nel registro EDI)

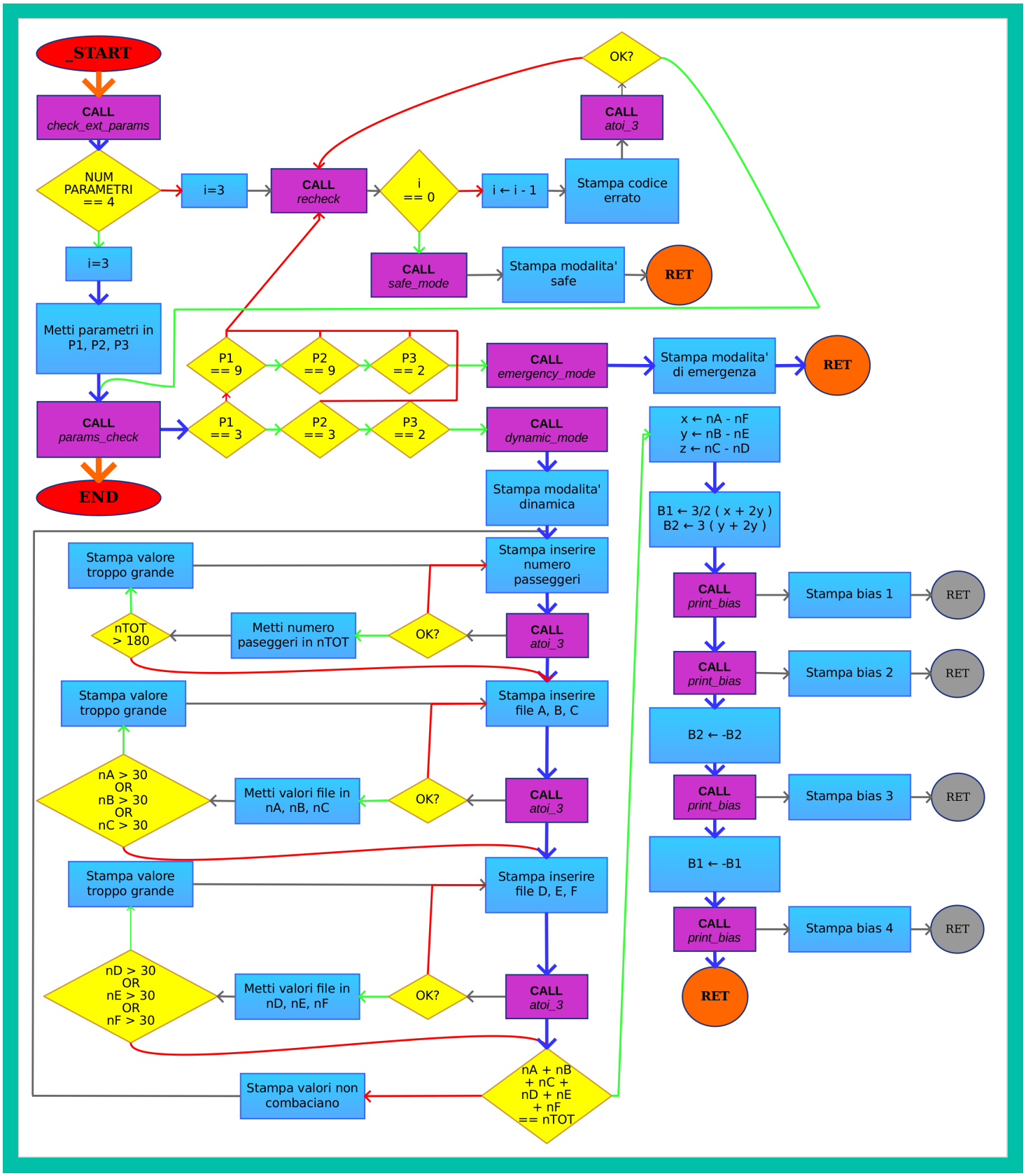
getch:
→ la funzione non riceve parametri
← la funzione passa nel registro EBX il carattere appena letto dal buffer di standard input

print_string:
→ la funzione riceve in EAX l'indirizzo in memoria della stringa da stampare
← la funzione non passa alcun valore di ritorno

itoa:
→ la funzione riceve in EAX un numero intero (negativo o positivo) da stampare
← la funzione non passa alcun valore di ritorno

Mario Alessandro Barbara (VR 387794)

DIAGRAMMA DI FLUSSO



Mario Alessandro Barbara (VR 387794)

DESCRIZIONE SCELTE PROGETTUALI

Modularita': La scelta progettuale veramente fondamentale in un programma e' quella del suo design. Per questo motivo ho deciso di rendere il programma il piu' modulare possibile, in maniera tale che risultasse essere facilmente manipolabile. Questo fattore si puo' notare da vari elementi del programma: la funzione _start che e' quasi vuota, l'utilizzo di molte funzioni esterne, funzioni dedicate ad ognuna delle 3 modalita' (in modo da controllarne il flusso e da focalizzare l'attenzione sullo scopo centrale del programma), funzioni esterne per la lettura da tastiera e stampa su video molto flessibile (la funzione di lettura, in particolare, e' piuttosto resistente all'inserimento di input errati). Le funzioni sono state riutilizzate il piu' possibile (la "*atoi_3*", per esempio, e' in grado di ricevere in input 1 o 3 parametri, siano essi cifre o numeri molto grandi o altro).

Input dei parametri: Durante lo svolgimento di questo elaborato, si e' molto riflettuto sull'inserimento dei parametri. Si è cercato di attenersi il piu' possibile alle specifiche, progettando tuttavia un programma che fosse il piu' flessibile possibile. Per esempio, l'input dei 3 parametri durante l'esecuzione programma segue un protocollo ben preciso:
"numero_spazio_numero_spazio_numero_invio". Cio' significa che qualunque input che non dovesse essere conforme a tale regola, dovra' essere individuato e scartato.
Al contempo, le specifiche sono ben precise sul passaggio esterno di parametri al programma: verranno inserite 3 cifre, separate da spazi. Dunque non e' stata applicata la regola precedente durante il controllo dei parametri esterni al programma.
Un altro fattore di mio interesse erano i valori che potevano essere accettati negli input corretti. 180 e' il massimo dei passeggeri dell'Airbus, come da specifiche, ma cosa fare per quanto riguarda il numero di passeggeri per ogni fila? Osservando attentamente la figura dell'Airbus nelle specifiche, non e' stato possibile capire esattamente quanti passeggeri ogni fila potesse accettare (il numero di posti totale non e' nemmeno 180), tuttavia era chiaro che essi dovessero essere uguali per ogni fila. Dunque, 180/6 = 30 passeggeri per ogni fila.

Utilizzo dei registri: Si e' scelto di utilizzare i registri per il passaggio di valori durante l'esecuzione del programma principalmente per 2 motivi. Primo, in un progetto cosi' ristretto come quello di questo programma, risulta piuttosto facile utilizzare i soli registri per il passaggio di parametri e ridurre il numero di righe di codice scritte.
Secondo, ritengo essere buona pratica effettuare il minimo di accessi a memoria possibile all'interno di un programma, poiche' spesso e' il fattore principale di un eventuale rallentamento, anche se questo programma e' sufficiente piccolo affinche' questo non risulti essere un problema.
Dovendo utilizzare solamente I registri per effettuare tutti i calcoli (con l'ausilio di alcune istruzioni *push* e *pop*), 4 risultava essere un numero troppo piccolo a volte. Per questo, dopo un po' di ricerca, ho scoperto altri 2 registri dei processori Intel x86 che potevo utilizzare alla pari dei registri general purpose (essi sono usati durante l'esecuzione di funzioni particolari del processore per lo spostamento di grossi dati dalla memoria): ESI ed EDI.

Comandi speciali in asm: Durante lo svolgimento del programma ho deciso di utilizzare alcune procedure un po' speciali. Una e' l'utilizzo delle istruzioni "*pushal*" e "*popal*": queste istruzioni salvano tutti i registri sullo stack e li ritirano e sono state utili nelle funzioni piu' generiche e modulari (come la "*print_string*") per garantire la corretta esecuzione del programma.
Un'altra tecnica utilizzata e' stata la sanitizzazione dei frame di ogni funzione nello stack: si salvavano i

valori dei registri EBP ed ESP ad ogni inizio funzione, e alla fine si ripristinavano. Questa tecnica e' stata utile per evitare segmentation fault dovuti alla impossibilita' di gestire il numero di "*push*" sullo stack durante l'esecuzione di alcune funzioni (come la "*atoi_3*", per esempio).
Un'ultima istruzione speciale utilizzata e' stata "cdq". E' stata di gran aiuto per la divisione con segno dei registri a 32 bit, poiche' estendeva il segno del registro EAX nel registro EDX, preparandolo dunque per la divisione con segno per valori di tipo long.